Les Cahiers du GERAD ISSN: 0711–2440

Deadlock Avoidance and Detection in Railway Simulation Systems

B. Simon, B. Jaumard,
T.H. Le

G–2013–43

June 2013

Les textes publiés dans la série des rapports de recherche HEC n'engagent que la responsabilité de leurs auteurs. La publication de ces rapports de recherche bénéficie d'une subvention du Fonds de recherche du Québec – Nature et technologies.

Deadlock Avoidance and Detection in Railway Simulation Systems

Bertrand Simon

École Normale Supérieure
Lyon, France

bertrand.simon@ens-lyon.fr

Brigitte Jaumard
Thai Hoa Le

GERAD & Department of Computer Science and Software Engineering
Concordia University
Montréal (Québec) Canada

bjaumard@cse.concordia.ca

June 2013

Les Cahiers du GERAD

G–2013–43

Copyright © 2013 GERAD

Abstract: Avoiding or preventing deadlocks in simulation tools for train scheduling remains a critical issue, especially when combined with the objective of minimizing, e.g., the travel times of the trains. In this paper, we revisit the deadlock avoidance and detection problem, and propose a new deadlock avoidance algorithm, called DEADAALG, based on a resource reservation mechanism. The DEADAALG algorithm is proved to be finite, and either detects an unavoidable deadlock resulting form the input data or provide a train scheduling thanks the SIMTRAS algorithm, which is free of deadlocks in $O(|S|.|T|^2 \log |T|)$, where T is the set of trains and S is the set of sections in the railway topology. Experiments are conducted on the Vancouver-Calgary single track corridor of Canadian Pacific. We then show that the SIMTRAS algorithm is very efficient and provides schedules of a quality that is comparable to those of an exact optimization algorithm, in tens of seconds for up to 30 trains/day over a planning period of 60 days.

Key Words: Deadlock Avoidance – Train Scheduling – Simulation – Deadlock Detection - Polynomial Algorithm

Acknowledgments: B. Jaumard was supported by NSERC (Natural Sciences and Engineering Research Council of Canada) and by a Concordia University Research Chair (Tier I) on the Optimization of Communication Networks.

The authors would like to thank Peter Finnie from CPR – Canadian Pacific Railway for his comments and for providing the data to validate the algorithms presented in this work.

1 Introduction

While railway companies are still using controllers for the real-time management of their trains, they also use simulation tools in order to mimic as closely as possible their daily operations in order to better understand the delays and better plan the train schedules so as to minimize the travel times (freight trains) or the tardiness (all trains). However, simulation tools still lack efficient tools in order to detect and avoid deadlocks, and provide meaningful results on a network operated under conditions close to its full network capacity, i.e., under traffic conditions that do not hamper significantly the travel times due to many train meets with waiting times.

A deadlock is a situation in a resource allocation system in which two or more processes are in a simultaneous wait state, each one waiting for one of the others to release a resource before it can proceed. Deadlock detection and avoidance has been studied not only for train systems, but for different types of resource systems, e.g., [12]. While it is now a well solved problem in the context of resources and processes where pre-emption is possible, it remains a poorly solved problem in the context of trains where train pre-emption does not make sense, and where the dynamic location of trains (the equivalent of processes in a computer system) makes the deadlock detection and avoidance more complex.

Current practice in existing simulation tools [3, 11] is to use a myopic look ahead test to avoid deadlocks, i.e., allows a train to move on the next segment if there will be no deadlock in the next 2 or 3 segments. With medium or high train densities, such a myopic vision is not enough in order to avoid deadlocks.

In this paper, we propose an original DEADAALG algorithm for deadlock detection and avoidance which very significantly improves on the classical Banker's and the Banker's like algorithms, as it is based on a track section reservation mechanism of minimum length. In addition, the DEADAALG algorithm has a $O(|S|.|T|)$ complexity, where T is the set of trains to be scheduled and S is the set of sections in the railway topology. It is therefore a highly scalable algorithm, which can easily be embedded in a train scheduling algorithm, called SIMTRAS algorithm, in the context of the design of a simulation tool.

The paper is organized as follows. In Section 2, we review the most recent results on deadlock avoidance in railways systems. In Section 3, we discuss deadlock avoidance and detection, as well as a basic train scheduling in the context of the design of a simulator. The newly proposed deadlock avoidance algorithm is detailed in Section 4, with the proof of its correctness and complexity. A more efficient scheduling algorithm, the SIMTRAS algorithm, that is deduced from the DEADAALG algorithm is described in Section 5. Numerical results are presented in Section 6 on several data set instances in order to evaluate the performance of the SIMTRAS algorithm and its performance comparison with an exact optimization algorithm for train scheduling. Conclusions are drawn in the last section.

2 Literature Review

While there has been many papers discussing deadlock prevention, detection and avoidance for computer systems, in which pre-emption is usually an option in order to break a deadlock, this is not the case for railway systems. There is still a need today for better deadlock detection and avoidance algorithms in order to design and develop efficient simulation tools for railway operations on single track or mesh railway networks.

The most cited Banker's algorithm [4] (or in the Appendix A of [1]), as well as its modifications, see, e.g., [1, 2, 3] are not well adapted to train scheduling, as the algorithms do not guarantee a strategy for deadlock avoidance with an efficient resource reservation mechanism, and requires multiple resource allocation searches without any guarantee to be able to generate a train scheduling with deadlock avoidance when such a train scheduling exists.

More recently, in [9], Pachl proposed a new deadlock avoidance method, called Dynamic Route Reservation (DRR). The key idea is to consider that when a train requests lo leave the section it is running on and move onto the next one, we must check, fast enough, that this movement is safe before allowing it, and decide how the system must evolve before and after this movement. The DDR method uses a reservation process

that succeeds if the move is allowed, and fails otherwise. The limitation of the DRR method is that, on the one hand, the reservation process can fail even if the initial state is safe (i.e., there exists a train scheduling without any deadlock), and on the other hand, the process can succeed while the train movement implies an initially avoidable deadlock. Indeed, the process contains arbitrary choices at some iteration of the DDR algorithm, which can lead to a false diagnosis of unavoidable deadlock.

Then, in [10], Pachl modified its reservation process so that if the initial state is solvable, the reservation succeeds. However, the new reservation process has a critical drawback: no deadlock detection is included in the algorithm, consequently, when a deadlock occurs, the behaviour of the algorithm is not defined. Furthermore, a reservation can be confirmed while later it leads to a deadlock that could have been avoided, and the deadlock issues with arbitrary choices at some iterations remains. An illustration is given in Figure 1. Therein, the green train, initially on section 04 should not be allowed to travel on section 02 (see why in Section 4.1 with Rule 6), however, Pachl's algorithm rules authorize it.

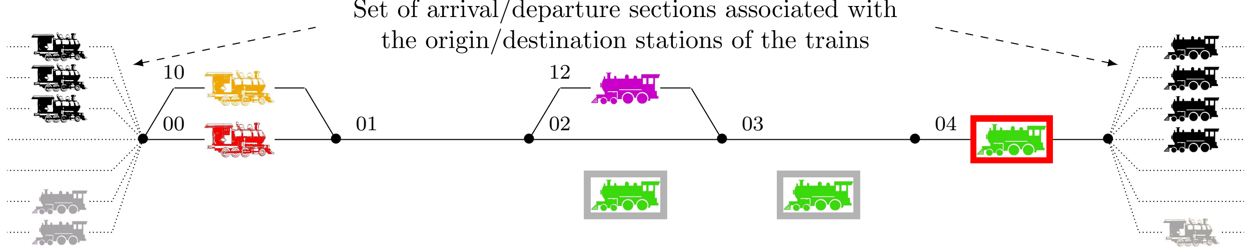


Figure 1: An illustration of the drawback of Pach's method [10].

Our contribution is, on the one hand hand, to modify the reservation process so as to make the algorithm (existence of a deadlock) deterministic, and, on the other hand, deduce a train scheduling that is deadlock free in polynomial time, whenever one such schedule exists.

3 Deadlock Avoidance/Detection and Train Scheduling Simulators

We consider a rail system consisting of a single line, with a single two-way track between stations or sidings. Each track is divided into segments which are separated by sidings or stations, and each segment is divided into a set of sections. Tracks can be used by trains running in both directions, and trains can meet and pass at stations or sidings. Sidings are typically added to a railway line in order to allow two trains to pass one another and are the most common method used to expand rail network capacity. Sidings are typically built long enough to permit trains to come to a full stop inside the siding while remaining clear of the switches at either end. In this paper, we assume that sidings are not overlapping. We define the set of sections as the set of segment sections and siding tracks, including the departure/arrival sections as illustrated in Figure 1. It is denoted by S, and indexed by s.

The set of trains can be described by a set of Eastbound and Westbound trains, denoted by T and indexed by t, with departure time $\text{DEPART}_s(t)$ at the origin of their departure section. Two trains in opposite directions are not allowed to be on the same track segment and they can meet each other only at a siding or a station. Two trains in the same direction can be running on a segment at the same time but they must maintain a safety distance, and they can pass each other only at a siding or a station.

The output of the train scheduling problem is either a deadlock if no feasible scheduling can be found for all the trains, or a train scheduling with the arrival/departure times of all trains at each siding/station. Travel times are deduced from the arrival times at the destination stations.

Train scheduling simulators divide into two categories: event-driven (or synchronous) (e.g., [5]) and time driven (asynchronous) (e.g., [8] simulators). Event-driven simulators works quite similar to scheduling: for a given set of trains, the sum of the overall waiting times for each train corresponds to the overall required delays

in order to get a train scheduling without any deadlock. In addition, stochastic delays may be generated in order to model unforeseen events, which often arise in practice. In such a case, the simulator generates a disturbed train scheduling. Whether time or event driven, simulators all face deadlock issues, and sometimes use scheduling data in order to circumvent them. We next propose a new deadlock avoidance/detection algorithm that can be easily adapted to both event and time driven train scheduling simulators.

4 A New Deadlock Avoidance/Detection Algorithm

We propose an original $O(|S|.|T|)$ deadlock avoidance/detection algorithm, called DEADAALG algorithm, which dynamically makes section reservation for the trains. If the DEADAALG algorithm succeeds with the completion of a sequence of successful reservations for all the trains until their final destinations, then a train scheduling without any deadlock can be deduced (see Section 5), otherwise, an unavoidable deadlock has been identified. The DEADAALG algorithm revisits the reservation process of the algorithm of Pachl [10]. The result is now an exact algorithm, which is rid of the arbitrary selections that potentially leads the algorithm of Pachl [10] to reach wrong conclusions (see the example of Section 2).

4.1 Reservation Process

In each section s, we define an ordered reservation list $\mathcal{L}_s$. Therein, we can insert a new reservation in any position, but always release first the reservation in the first position. At the outset, on section s, $\mathcal{L}_s$ is initialized with the unique train running on s, or is empty if there is no train on s.

A reservation is made for a pair (s, t) of a section and a train, and may have 4 different states:

$$\text{STATE_RES}_s(t) \in \{\mathsf{pending}, \mathsf{requested}, \mathsf{initiating}, \mathsf{confirmed}\} .$$

The reservations and their state evolve as follows over the iterations of the DEADAALG algorithm. At each iteration of the DEADAALG algorithm, reservations are in state requested on the sections on which the trains are waiting to move forward, see Figure 1, or on the sections on which the trains are running. Consequently, at each iteration, a train is selected (selection rules are discussed below), say t, and a reservation process is triggered on the (unique) section where there is a requested reservation for t. In such a case, the train reservation passes in the initiating state, and the reservation process keeps adding pending reservations for t, and may prompt reservation processes for other pairs (s', t') in a requested state. When no more reservation is triggered and no deadlock issue has been encountered, the reservation process is claimed successful and all the reservations added directly by the reservation process triggered by t, changes to confirmed state except the last one, which changes to requested state. It means there is a way out for train t up to this last section, and another reservation process will need to be triggered until the reservation process reaches the final destination of the train. Note that, at any time, there is at most one requested or initiating reservation per section. In the $\mathcal{L}_s$ reservation files associated with sections, the confirmed reservations are always placed at the beginning, and the pending ones at the end.

We say that train t is 'occupying' siding s, i.e., $t = \mathsf{occupying}(s)$, if t has an initiating or a requested reservation for s. If no train has such a reservation, then $\mathsf{occupying}(s)$ returns $\emptyset$.

The reservation process obeys the following rules:

1. If the current reservation is not for a siding section, the train must reserve a section ahead. Moreover, the reservation will be confirmed when booking ahead will be successful, meanwhile, the reservation has a pending state.
2. If a reservation is requested on section s, which does not contain any pending reservation, reservation is added in pending state, in the last position of the reservation file on s.
3. If a reservation is requested on a section which contains pending reservations, this last reservation (pending state) must be placed in front of the set of pending reservations, i.e, the latest reservation needs to be confirmed before the previous pending ones. Then, the associated train must reserve one section ahead, as there are reservations behind it. Note that this can only happen on segment sections, and is then enforced by Rule 1.

4. If a reservation is placed behind an initiating reservation, the reservation fails: it means there is a circular wait, i.e., a deadlock.
5. If the reservation is placed behind a requested reservation, the latter one launches a reservation process, and must successfully confirm it before continuing the process of the former train.
6. If there is a reservation request of train t for a siding section, proceed as follows. Let s_a and s_b be the two siding sections.

   a. There is an initiating reservation on siding section s_a: reserve on s_b.

   b. The siding is occupied by two trains running in the direction of t: the two sections are equivalent, so we can choose s_a. The train occupying s_b must immediately launch a reservation process. Then, because of Rule 5, the train occupying s_a must launch a reservation process for it. Reservation of t succeeds if the last two reservation processes are successful.

   c. The siding is occupied by two trains running in the direction opposite to t's: the two sections are equivalent, reserve one of them, say s_a.

   d. The siding is occupied by a train opposite to t on s_b and a train going in the same direction as t on s_a: reservation is made on the siding section of the train going in the same direction as t, s_a.

   e. The siding is occupied by one opposite train, on s_b: reservation is made on the free siding section, s_a.

   f. The siding is occupied by one train of the same direction as t, on s_a: reservation is made on the occupied section, s_a.

   g. The siding is not occupied: proceed with a reservation on any of the two, say s_a.

4.2 DEADAALG Algorithm

We next describe in detail the DEADAALG algorithm that, thanks to the rules described in the previous section, determines whether there exists a feasible schedule without any deadlock.

When the reservation process succeeds, each train 'occupies' exactly one section. The reservation process is relaunched, assuming that the trains are positioned in the section they 'occupy', until all the trains have successfully reached their final destination, or a deadlock has been encountered.

Train selection influences the average train travel times, but not the detection of a deadlock, as will be shown in the correctness proof of the DEADAALG algorithm.

The DEADAALG algorithm builds two reservation lists, a global one, $\mathcal{L}_s$ (see its definition in Section 4.1), and $\mathcal{L}_t^{\text{process}}$ that is local to RESERVATION(t). It calls two functions, RESERVATION(t) and RESERVATION_SEC (t, s) that take care of the reservation of sections for t until either it ends successfully or it fails, and of the reservation of t on section s, respectively.

Algorithm DEADAALG

Require: A bidirectional single track network, its set of sections and a set of trains with its train departure plan.

Ensure: Determine whether there exists a feasible schedule without any deadlock.

  $\mathcal{L}_s \leftarrow \emptyset$ for all $s \in S$; STATE$_s(t) \leftarrow \emptyset$ for all $(s,t) \in S \times T$ ▷ Initialization

  $s \leftarrow$section on which t is ; STATE$_s(t) \leftarrow$ requested for all $t \in T$

  ▷ There can not be two trains with a requested state on the same section, i.e., two trains can not be on the same section

  while no deadlock has been detected or one train remains in the system **do** ▷ Core part

    Select a train t, that has not reached its destination ; RESERVATION(t)

An illustration of the DEADAALG algorithm is provided in Figure 2, where the Westbound black, purple, green and blue trains are occupying sections 16, 06, 04 and 14 respectively, while the Eastbound yellow, red and turquoise trains are occupying sections 12, 00 and 10 respectively. The reservation state is indicated with

Function RESERVATION(t)

Require: for all s, t : $\text{STATE}_s(t)$, $\mathcal{L}_s$; and a particular train t.

Ensure: Determine whether t can move without generating a deadlock. If yes, update the schedule with an elementary move of t. If no, fails.

$s \leftarrow$ unique section such that $\text{STATE}_s(t) =$ requested ; $s^{\text{init}} \leftarrow s$

$\text{STATE}_s(t) \leftarrow$ INITIATING ; next_sec(s): returns the section after s on the route of t.

$\mathcal{L}_t^{\text{process}} \leftarrow \{s\}$; $s \leftarrow$ next_sec(s)

while s is not a siding track **do** ▷ Rule 1

RESERVATION_SEC (t, s) ; $\mathcal{L}_t^{\text{process}} \leftarrow \mathcal{L}_t^{\text{process}} \cup \{s\}$; $s \leftarrow$ next_sec(s)

if the endpoint of s is the final destination of t **then**

$\text{STATE}_{s'}(t) \leftarrow$ confirmed for all $s' \in \mathcal{L}_t^{\text{process}}$; Terminate function RESERVATION(t)

▷ s is a siding track. Let s_1 and s_2 be the two sections of it.

$t_1 \leftarrow$ occupying(s_1) ; $t_2 \leftarrow$ occupying(s_2)

Map s_1, s_2 to s_a, s_b according to Rule 6, and let s^{selected} be s_a

case 1: Apply Rule 6b with s^{selected} $\rightsquigarrow$ RESERVATION (t_b) ; RESERVATION_SEC (t, s^{selected})

case 2: Apply any of the other rules with s^{selected} $\rightsquigarrow$ RESERVATION_SEC (t, s^{selected})

$\text{STATE}_s(t) \leftarrow$ confirmed for all $s \in \mathcal{L}_t^{\text{process}}$; $\text{STATE}_{s^{\text{selected}}}(t) \leftarrow$ requested

a letter next to each train, on the left or or to the right depending on whether the train travels Westbound or Eastbound. Reservation is first sought for the purple train, and we assume without loss of generality that it chooses section 04. Let us assume that, contrary to the rules of the DEADAALG algorithm, that the blue train does not immediately trigger a reservation process. Then, the green train has to launch one, and leads to the situation described in the top of Figure 2. The green train successfully completes its reservation, but then we reach a deadlock, as there is no solution with the purple train moving in section 04, and the green train moving in section 03 before any move of the yellow train.

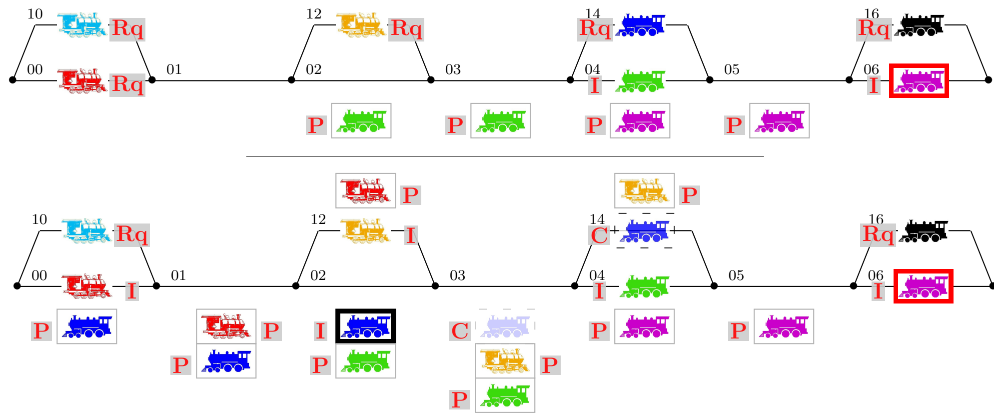


Figure 2: An illustration of the DEADAALG algorithm.

The failure of previous reservation process is the motivation of Rule 6b that forces the blue train to reserve before the green one. Then, the blue train can successfully complete its reservation up to section 02, the reservations change to confirmed in sections 03 and 14 and the reservation changes to requested in section 02. Then, the green train reservation is entailed, and proceeds with a reservation in section 02, and then causes the blue train to reserve until section 00. At this point, the red train reserves on section 01 and its reservation is then placed before the pending one on the blue train. Then, this causes the yellow train reservation, and

Function RESERVATION_SEC (t, s)

Require: Train t ; Section s and its reservation list $\mathcal{L}_s$; For all $(s,t) \in S \times T$: $\text{STATE}_s(t)$; For all $s \in S$: $\mathcal{L}_s$.

Ensure: Determine whether t can successfully reserve s according to the reservation rules. If no, an unavoidable deadlock has been identified. If yes, place the reservation and triggered the additional reservations entailed by a successful reservation for t.

$t' \leftarrow$ Occupying(s) ; $i \leftarrow$ position of the first pending reservation in $\mathcal{L}_s$; $\text{STATE}_s(t) \leftarrow$ PENDING
▷ all the reservations in $\mathcal{L}_s$ placed after i are pending
if $t' \neq \emptyset$ and $\text{STATE}_s(t') =$ Initiating **then** FAIL ▷ Rule 4
else if there is no pending reservation in $\mathcal{L}_s$ **then** ▷ Rule 2
Add t to the end of $\mathcal{L}_s$; **If** $t' \neq \emptyset$ **then** RESERVATION (t') ▷ Rule 5
else Insert t in position i of $\mathcal{L}_s$ ▷ Rule 3
▷ In this case, the section is on a segment, and so the RESERVATION process continues on t

is located between the blue confirmed and the green pending reservations on section 03, and concludes on section 14, which is not occupied (second scheme). Finally, all reservations succeed.

Illustration of some of the cases in provided in Appendix A, together with a complete sequence of the DEADAALG in Appendix C.

Note that it is easy to derive a train scheduling from the DEADAALG algorithm, in the context of an event driven simulation tool, using the following rule:

Scheduling Rule 1 *At any time, train t receives permission to move to another section only if t has a confirmed reservation in the first position of the reservation file on this section. Then, the reservation of the former section (which was also confirmed, and in first position) is released and suppressed. If the endpoint of the new section is the final destination of t, it has successfully reached it.*

4.3 DEADAALG Algorithm Performance and Complexity

Due to the lack of space, we provide below an outline of the correctness and complexity of the DEADAALG algorithm in the case of sidings with 2 alternate tracks, and refer the reader to Appendix B for the detailed proof.

Theorem 1 *The DEADAALG algorithm is finite. With an $O(|S|.|T|)$ complexity, the DEADAALG algorithm concludes that at least one deadlock cannot be avoided, or exhibits a train scheduling free of any deadlock on a single track railway system.*

The initial locations of the trains together with their departure times, i.e., the train departure plan, can be arbitrary under the condition that there is at most one train per section, and the destinations of the trains correspond to the endpoints of the line network, see Figure 1. A train departure plan is solvable if all the trains can reach their destination without encountering any deadlock. Within the DEADAALG algorithm, it means that we can reach a train configuration in which all the reservations are in the confirmed state except on one track section of its destination.

We first focus on the RESERVATION function. At each iteration, the train configuration which is output by the RESERVATION function is such that each train has a unique requested reservation state and this last reservation is for the section it is lying on. The first step of the proof consists in demonstrating that if RESERVATION(t) is launched on a solvable train configuration and concludes, then it is possible to reach a train configuration that is output by RESERVATION(t) throughout iterative applications of Scheduling Rule 1 (see end of Section 4.2), and this output train configuration is solvable. Such a demonstration can be made in three steps:

Step 1. If a train departure plan is solvable, there exists a solution such that a train never enters a siding if the other section of this siding is occupied by a train in the same direction (used for Steps 3, 4 and 5).

Step 2. For all train configurations, if RESERVATION succeeds, then its output train configuration is valid (i.e., at most one train per section, and each train is on one section) and reachable from the input train configuration, repetitively using Scheduling Rule 1.

Step 3. If a RESERVATION succeeds, then, on the set of spanned rail track sections, in the output train configuration, for each direction, there is a sequence of sections without any train in the opposite direction, going through the latter rail track.

Once those three demonstration steps are completed, we can conclude, based on Step 3, that the output train configuration is solvable. *This way, we do not focus on the trains involved in the* RESERVATION *function call, but only on the set of spanned track sections, and we prove it to be in a safe situation, i.e.,* RESERVATION *is a safe modifier of the train configurations, which do not create deadlocks.*

It remains to show that RESERVATION does not fail on a solvable train configuration. This is done in 2 steps. From now on, we consider the function MODIF, which forgets Rule 6, and allows the user, at each siding, to choose the section the train should try to reserve. Then, as soon as there is a failure, the MODIF function fails without trying any other choice.

Step 4. If RESERVATION fails, all the choices in the MODIF function would also fail.

Step 5. Assume that a function (either MODIF or RESERVATION) fails, and did not launch successfully other functions. Then, there is no solution in which each train can go through all the sections it has tried to reserve, while satisfying the rule of Step 1.

Once these demonstration steps are completed, we can conclude that if RESERVATION fails, all the MODIF function calls fail, and we can come down to the case of Step 5 (using the previous result) to show that all choices lead to a deadlock, so the input train configuration was not solvable. *This way, we show that the choices made in* RESERVATION *are relevant: if there is a solution, it finds one.*
The DEADAALG algorithm works as follows. While there is a train in the system, we choose one (the order will only modify the associated scheduling, see Section 5), and call RESERVATION on it. If it succeeds, we continue with the new train configuration. Otherwise, we claim that the train departure plan was a deadlock.

Proof of the complexity. We cannot call twice RESERVATION_SEC(t, s) on the same pair (t, s), so it cannot be called more than $|S|.|T|$ times. If we implement $\mathcal{L}_s$ using double linked lists and keep pointers to the last train in the confirmed state in each list, we get a complexity of $O(|S|.|T|)$. □

Proof of the correction. If the train departure plan is solvable, the first call to RESERVATION succeeds and leads to a solvable train configuration. Then, using induction, all the RESERVATION calls succeed and lead to solvable train configurations. Using Scheduling Rule 1, we can exhibit a scheduling free of any deadlock. If the train departure plan is not solvable, when DEADAALG concludes, trains must remain in the system, so the reservation process has failed, which indeed means that there is no solution without encountering a deadlock. □

5 Train Scheduling Simulation with Deadlock Avoidance/Detection

We explain earlier how to derive a first train scheduling from the output of DEADAALG algorithm, which has been proved to be correct, thanks to Theorem 1. We now discuss how to improve it with respect to the minimization of the average travel times of the trains.

While applying DEADAALG, the departure time of train t on section s, denoted by $\text{DEPART}_s(t)$, can be computed in polynomial time, using the section travel times. It is computed in order to prevent unnecessary stops on a segment, by forcing trains to wait on the sidings if they can not reach the next one without stopping. At the beginning of a RESERVATION(t) call, trains may have to wait on their departure track (that either belongs to a siding or to a segment).

Assuming the objective is to minimize the average travel times of the trains, at each step, we select train t with the smallest $\text{DEPART}_s(t)$ on the s section it is requesting. This way, the repartition of the requested

Algorithm SIMTRAS

Require: Set of trains with their departure time and their route toward destination.
Ensure: Produce a train schedule that is free of deadlocks, if possible. If not, exhibit an unavoidable deadlock.

Associate the segment and siding sections with their travelling time
Add multiple parallel sections at the origin/destination stations of the trains
$\mathcal{L}_s \leftarrow \emptyset$ for $s \in S$; $\text{STATE}_s(t) \leftarrow \emptyset$; $\text{DEPART}_s(t) \leftarrow \emptyset$ for $(s,t) \in S \times T$
for $t \in T$ **do**
  $s \leftarrow$ departure section of t (one extremity)
  $\text{STATE}_s(t) \leftarrow$ requested ; $\text{DEPART}_s(t) \leftarrow$ departure time of train t

while there is no failure or one train remains in the system **do** ▷ Core loop
  Select t such that: $\forall t', \text{DEPART}_{s(t)}(t) \leq \text{DEPART}_{s(t')}(t')$, where $t = \text{occupying}(s(t))$
    ▷ if t is on a siding track, and a reservation of an opposite train prevents him from leaving at $\text{DEPART}_{s(t)}(t)$, we associate the minimal time at which it can leave instead of $\text{DEPART}_{s(t)}(t)$
    ▷ Ties can be broken by, e.g., selecting the most eastern siding s, and then for remaining ties, with the selection of eastbound train
  RESERVATION(t)
    ▷ We repeat the reservation until t has passed the section of the first train of the queue
  if this reservation has directly launched n other reservations because of rule 5 **then**
    Repeat n times: RESERVATION(t)
return the schedule: Each train t leaves section s at time $\text{DEPART}_s(t)$

▷ To be added at the end of RESERVATION
$\text{DEPART}_{s^{\text{init}}}(t) \leftarrow \min\left\{\tau \geq \text{DEPART}_{s^{\text{init}}}(t) \ : \ t \text{ can leave } s^{\text{init}} \text{ at } \tau \text{ and reach } s^{\text{selected}} \text{ with no stop}\right\}$
for $s \in \mathcal{L}_t^{\text{process}} \cup \{s^{\text{selected}}\}$ **do**
  $\text{DEPART}_s(t) \leftarrow \text{DEPART}_{s^{\text{init}}}(t) +$ travel time from the end of s^{init} to the end of s
  Change the position of t in $\mathcal{L}_s$ so that the list remains ordered by increasing $\text{DEPART}_s(t)$

sections at each step is the closest to a snapshot of the future scheduling, and it behaves as a greedy algorithm: at each step, the first train that arrived at a segment has the highest priority to take it. When a requested reservation is on siding track s for train t, and a reservation for a train in the opposite direction prevents it from leaving at $\text{Depart}_s(t)$, we use the minimal time at which it can leave instead of $\text{Depart}_s(t)$. This way, the priority is left to the direction of the first train crossing the segments, and it reduces the queues at the bottleneck sidings. This adds a $\log |T|$ factor to the complexity.

In addition, to avoid the formation of queues due to congestion, we added the following feature. If the reservation has directly led to n other reservations, because of Rule 5, the process is reapplied n times on the same train. In this way, when we encounter a queue, the trains move so that the last train passes the initial section of the first train. Then, when there is a bottleneck, all trains move in one block, and we avoid the case of one train, in each direction, that monopolizes the bottleneck alternatively.

The last modification added is to check if the train can be scheduled before other trains with confirmed reservations. Indeed, if, because of queues, train t has successfully reserved a section, but there is another train, say t', that can pass through that section before t, t' should not be held back. Therefore, at the end of the Reservation function, we check if the reservations can be placed sooner in the lists. The drawback of this modification is to increase the time complexity of the algorithm, because we have to scan a not-constant part of the $\mathcal{L}_s$ to find the minimum possible time. A basic implementation leads to a $O\left(|S||T|^2 \log |T|\right)$ complexity.

In order to process segment data with travel times that vary and depend on the direction (Est vs. West), we modelled each segment by a section. We allow two trains, going in the same direction, to be initially on the same section if the differences of the departure times are at least the safety time. In this way, the

algorithm behaves as if the segments contain various sections, but the precision of the position of the trains is more accurate.

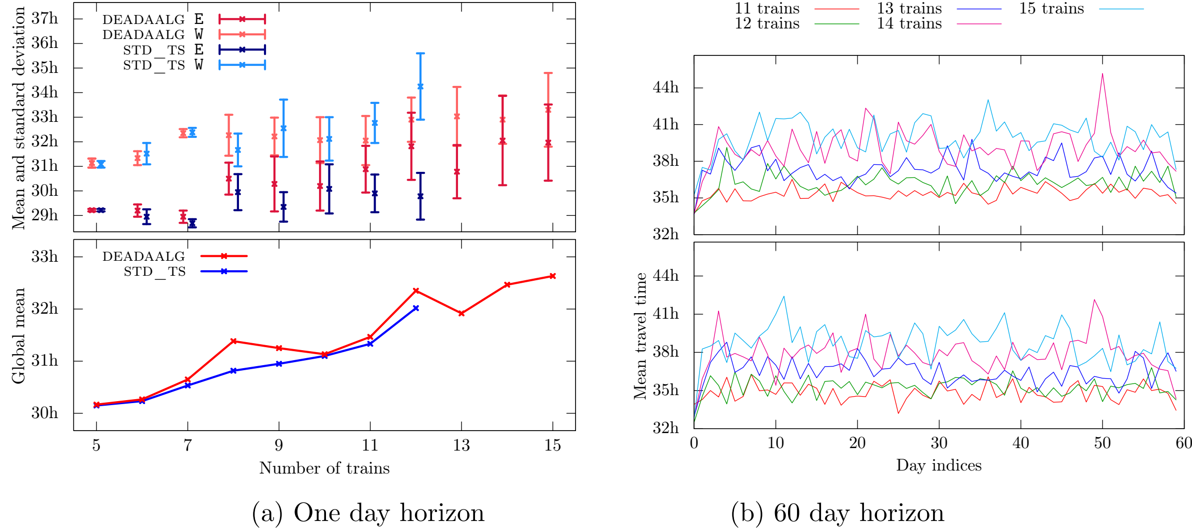
6 Numerical Experiments

All the algorithms and functions described in the previous section were implemented in C++, as part of a event driven simulation tool.

We evaluated the performance of the DEADAALG and SIMTRAS algorithms on the Canadian Pacific Railway (CPR) network between Calgary and Vancouver [6], i.e., the most busy corridor of the CPR network. It is a single track railway system, with 77 sidings/stations. In terms of capacity (number of alternate tracks), we assume 1 alternate track at every station/siding.

We use a set of a 8 to 30 trains, with the same number of trains from Vancouver towards Calgary as from Calgary towards Vancouver. Departure times are uniformly distributed over a time period of 24 hours when considering a 24h planning period, and over a time period of $24(1 - \frac{1}{|T|})$ hours for planning periods spanning several days (60 in our experiments). Consequently, when the number of trains increases, their departure times are less spaced out.

The first observation is that the SIMTRAS algorithm is highly scalable: the computing time on 77 segments is about 10 seconds for 1,000 trains, and about 40 seconds for 2,000 trains.



(a) One day horizon (b) 60 day horizon

Figure 3: Average travel times.

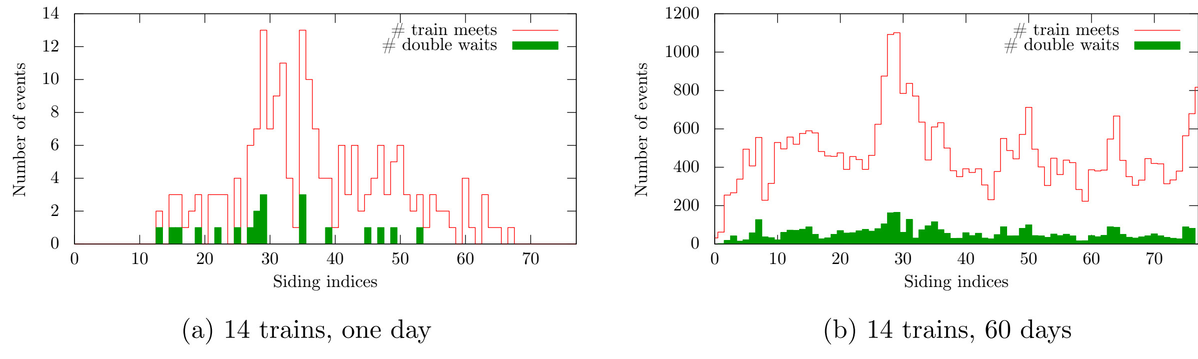
In Figure 3, we plotted the average travel times of the overall daily fleet of trains obtained with the SIMTRAS algorithm, for different number of daily trains (from 5 to 15 over a one day time period in Figure 3(a) and from 11 to 15 over a 60 day time period in Figure 3(b)), and their standard deviations. We distinguish the Eastbound (lower $\updownarrow$ in Figure 3(a) and lower diagram in Figure 3(b)) from the Westbound trains (upper $\updownarrow$ in Figure 3(a) and upper diagram in Figure 3(b)).

In addition, in order to evaluate the quality of the train timetables output by the SIMTRAS algorithm, we added the results obtained by the ϵ-optimal SDT_TS algorithm of Jaumard *et al.* [7], see the blue (darker) intervals. As for the SIMTRAS algorithm, the blue intervals are centered around the train scheduling output by the SDT_TS algorithm.

Taking into account the standard deviations measured over a one day time period, we observe that the average travel times are fairly stable over a 60 day time period, i.e., there is no deterioration of the average travel times over the time. Longer travel times for the West bound trains is due to the average higher load of the Westbound trains in comparison with the Eastbound trains in the CPR Vancouver/Calgary corridor.

In Figure 3(a), we observe that the standard deviations of the SIMTRAS and SDT_TS algorithms are fairly similar, and we verify that the results of the SIMTRAS algorithm are lower bounds on the SDT_TS, taking into account the accuracy (ε) of the solutions: see the lower part of Figure 3(a) as the accuracy of the SDT_TS algorithm is on the overall average travel times, in which average times on the Eastbound and Westbound trains are not distinguished. Differences between the solutions of the two algorithms does not increase with the number of trains, and we believe it is one of the first times or the first time, that such differences are measured. Note that no schedule information is integrated in the SIMTRAS algorithm, which has been implemented as a "pure" simulation algorithm with a heuristic rule for the next train to be selected in the reservation process in the core loop, see the detailed description of the SIMTRAS algorithm in Section 5.

In Figure 4, we investigate the siding usage firstly over a one day period in Figure 4(a) and then over a 60 day period in Figure 4(b). The red curve indicates the number of train meets. While over a one day period, sidings close to the origin or the destination are not used, we observe that the usage of the sidings is much more uniform over a 60 time period, although there are differences in their usage. The green curve represents the number of train meets associated with double waits, meaning that some trains need to wait on any of the tracks of the sidings, see the illustration in Figure 5. Therein, the blue train waits on the south track 04 of the siding while later the green train will need to wait on the north track 14 of the siding in order to avoid a deadlock.



(a) 14 trains, one day (b) 14 trains, 60 days

Figure 4: Siding usage.

Figure 5: A double wait that leads to deadlock avoidance.

7 Conclusion

We have proposed a first exact ($|S|.|T|^2 \log |T|$) and highly scalable deadlock detection and avoidance DEADAALG algorithm for train scheduling. In addition, the DEADAALG algorithm favorably competes with the SDT_TS exact algorithm of [7] for the minimization of travel times. and dominates all previously proposed algorithms for deadlock avoidance in the context of train scheduling. Future work will include generalizing the DEADAALG algorithm to sidings with more than 2 alternate tracks.

A Illustrations of the DEADAALG Algorithm

Glossary

Deadlock: a situation in a railway network where multiple trains are blocking each other such that neither train can continue, see Figure 6. A deadlock is always characterized by a circular-wait condition, in which the railway network produces closed circles of trains waiting for the release of tracks currently occupied by other trains.

Section: a part of the track large enough to contain a train, and such that only one train is authorized on it at any time.

Siding: a short stretch of railroad track used to enable trains on the same line to pass or to meet. In the present study, we assume that only one train can be on each section of the siding, and that the siding only contains one alternate track.

Segment: part of the track between two consecutive sidings, on which two trains in opposite directions can never run at the same time.

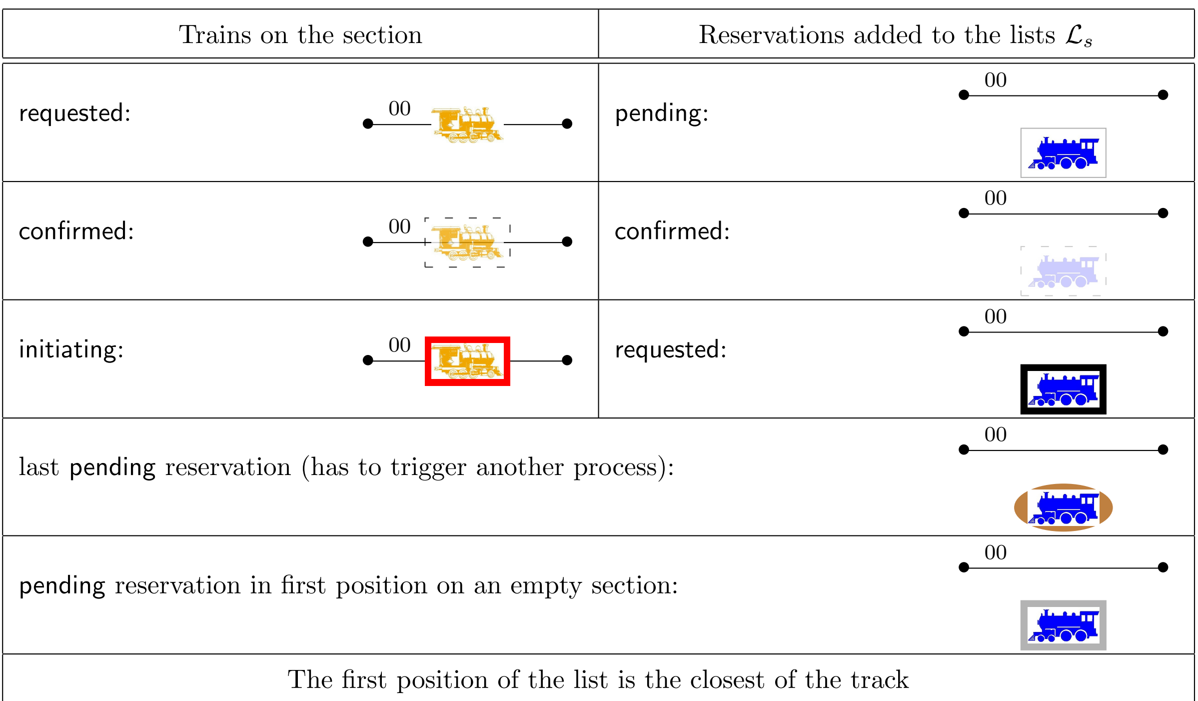
Train Configuration: an intermediate system, where each train is on a section.

Solution: a sequence of moves of the trains that allows them to reach their destination, consequently, without any deadlock. A train configuration that has a solution is called solvable.

Departure train plan: the outset configuration, i.e., the section on which each train starts, and the minimal time at which it can leave it.

List configuration: the output of RESERVATION(t), i.e., the lists $\mathcal{L}_s$, from which we can deduce the output train configuration of the RESERVATION(t) call.

Legend



Trains on the section	Reservations added to the lists $\mathcal{L}_s$
requested: 00	pending: 00
confirmed: 00	confirmed: 00
initiating: 00	requested: 00
last pending reservation (has to trigger another process): 00	
pending reservation in first position on an empty section: 00	
The first position of the list is the closest of the track	

The set of sections is denoted by S, and the set of trains is denoted by T.

Below are some illustrations of list configurations obtained during or after calls of RESERVATION(t) for various departure train plans.

Figure 6: A deadlock and a state that leads to a deadlock.

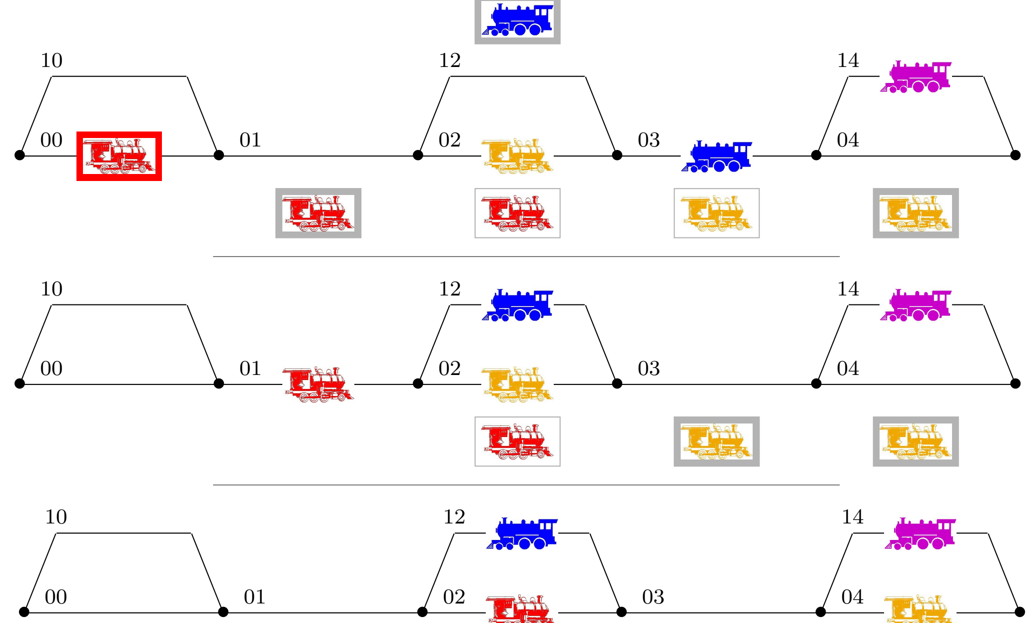


Figure 7: Example of reservations and simulation after the request of the red train. A train can move if it has reserved the next section in the first position, according to Scheduling Rule 1. Such reservations are indicated with a bold frame.

Figure 8: The red train requests to move. At the stage of the reservation process represented on the left diagram, the green train has to trigger a process (the red train is circled with brown because its reservation is not yet completed). So the reservation of the green train is placed in the first position of Section 01, by Rule 3 and leads to the right diagram.

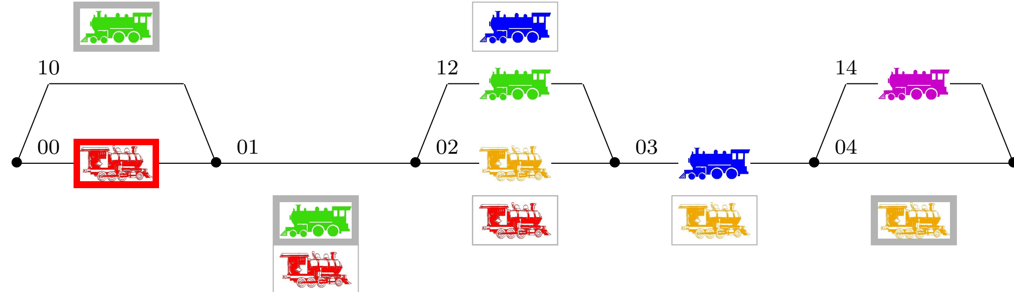


Figure 9: The red train requests to move. It entails the moves of the yellow, blue and green trains. Then the green train is placed in first position on Section 01, preventing the red one from moving immediately and consequently creates a deadlock.

Some limitations of Pachl's rules (cf. Section 2) are illustrated in Figures 10 and 11.

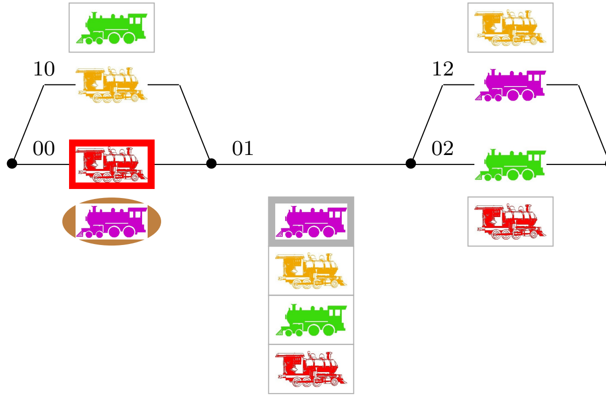


Figure 10: Such a circular wait can be reached, and nothing is designed to treat it in Pachl's rules, which motivated Rule 4.

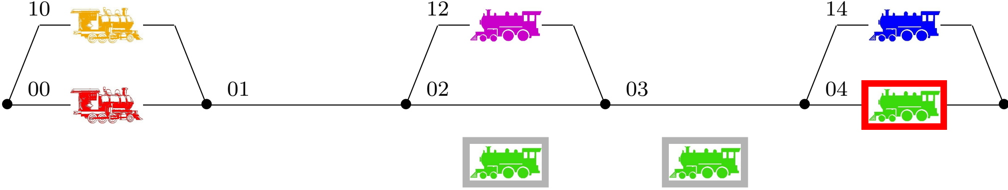


Figure 11: The green train should not move into Section 02, but Pachl's rules authorize it. See next figure for the behaviour of RESERVATION.

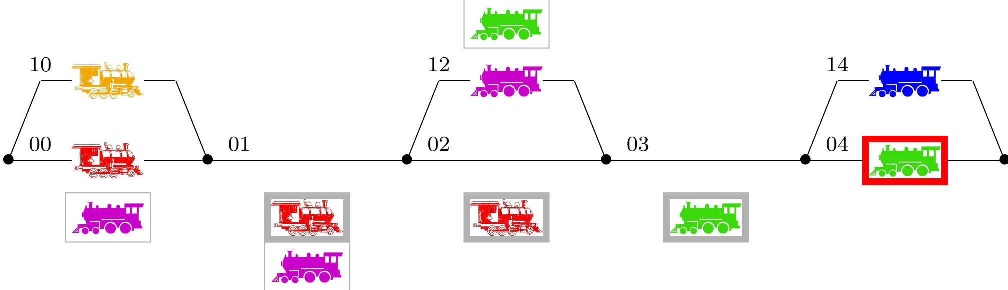


Figure 12: Correction of the failure of Pachl's algorithm as illustrated in Figure 11: If the green train requests section 12 instead, as enforced by Rule 6f, the purple train will trigger a reservation process, and then the red one will reserve on section 02, and will be placed first in section 01, so the reservation succeeds.

We provide below some more complex examples dealing with the advanced rules we have defined, and with the confirmed state. The trains that are framed by dashes represent the confirmed reservations, and the trains that are framed by black bold lines correspond to the requested reservations added by RESERVATION.

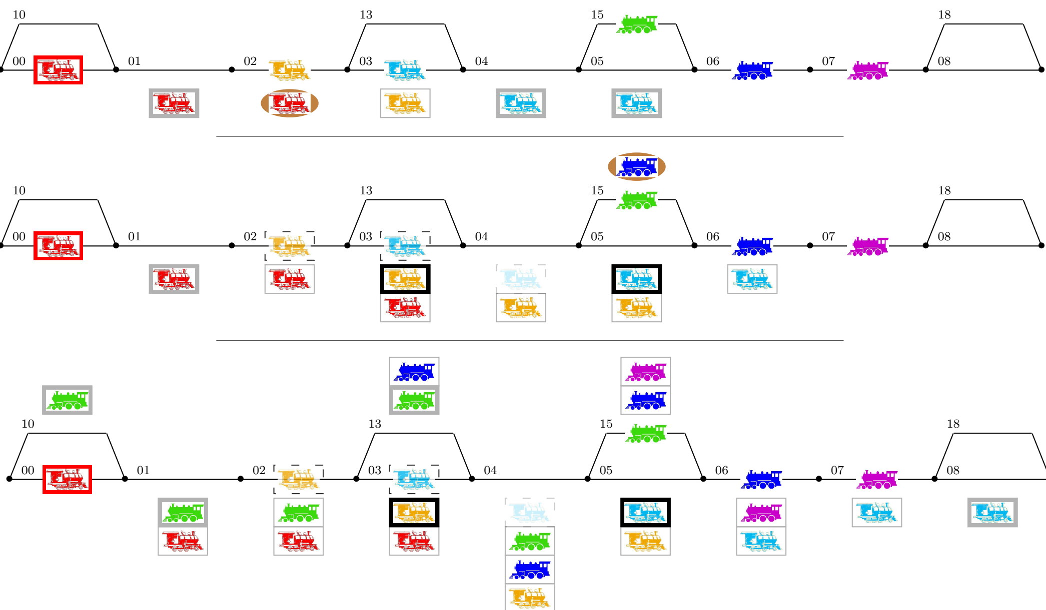


Figure 13: A tricky example, where the red train requests to leave. In the first list configuration (as represented in the top diagram), the yellow and cyan trains have successfully completed the reservations caused by the red one, so these reservations are now confirmed. In the second list configuration (middle diagram), the red train pursues its reservation on Section 03, and the function goes on. In the third list configuration (bottom diagram), note that the blue then the green train are placed in Section 04 between the cyan (confirmed) and the previous reservations (pending). Then, the reservation process succeeds.

B Proof of Theorem 1

We next detail the proof of Theorem 1.

The departure plan is arbitrary, with at most one train per section, and the destinations of the trains are at the endpoints of the line network. We associate the destination stations with a set of multiple tracks where the trains can stop, see Figure 1.

The proof of Theorem 1 relies on the RESERVATION function, is proved thanks to two theorems asserting that if the input configuration (which is arbitrary, not necessarily the departure train plan) is solvable, the RESERVATION call completes successfully and leads to a new solvable configuration.

Then, the correction of DEADAALG can easily be deduced. We next compute the complexity of the DEADAALG algorithm at the end of Appendix B.

As explained in the glossary, a train departure plan is solvable if all the trains can reach their destination without encountering any deadlock. Within the DEADAALG algorithm, it means that we can reach a train configuration in which all the reservations are in the confirmed state except on one track section of its destination.

The output configuration of a RESERVATION call is the configuration where the trains are all in the section in which they have the requested state, at the end of the RESERVATION call. The output configuration is strictly more advanced in time than the input configuration, as at least one train has triggered a reservation process, consequently the section in which it has the state requested in the output configuration is strictly ahead of the one in the input configuration.

We need a first lemma, which is next stated and proved.

B.1 Lemma 2

Lemma 2 *Given a section s and a train t, with s on the route of t, and until the end of the* DEADAALG *algorithm, if not stopped by a failure of the* RESERVATION *function,* $\text{STATE}_s(t)$ *successively equals to:*

- *requested, then initiating, then confirmed if s is the departure section of t. Otherwise,*
- *$\emptyset$, then pending, then confirmed if s is a segment section.*
- *$\emptyset$, then pending, then requested, then initiating, then confirmed if s is a siding section.*

Proof. If s is the departure section of t, $\text{STATE}_s(t)$ = requested at the outset. Then, when t launches a reservation process, $\text{STATE}_s(t)$ becomes initiating, then confirmed when this process succeeds.

Otherwise, if s is a siding section, $\text{STATE}_s(t) = \emptyset$ at the outset. When a reservation process reaches s, the state becomes pending. If the reservation does not succeed without reserving other sections for t, it means that the reservation of t has been placed in front of pending reservations on s. Then, there was a train in the initiating state on the siding when t reserved s, which is impossible. So the reservation ends successfully on this section and $\text{STATE}_s(t)$ changes to requested. Then, as in the previous case, it changes to state initiating then confirmed.

If s is a segment section, the behaviour is the same, except for the requested and initiating states. □

B.2 Theorem 3

We next need to prove the following intermediate theorem.

Theorem 3 *If* RESERVATION *ends successfully on a solvable input configuration, then the output configuration is reachable from the input configuration and is also solvable.*

To prove Theorem 3, we will use the following three lemmas.

Lemma 4 *If a train configuration is solvable, we can build a solution on which a train never enters a siding if the other track of this siding is occupied by a train going in the same direction.*

Proof. We say that an unstable event has occurred when a train enters a siding while the other track of this siding is occupied by a train going in the same direction.

Consider a solvable configuration, and one of its solution with the smallest number of such unstable events. We suppose that this number is not zero.

We consider the eastmost siding s on which such an unstable event occurs, and we will build a solution with one less occurrence of unstable events. Let W_{tracks} be the part of the track strictly at the west of s, and E_{tracks}, strictly at the east. The layout is represented Figure 14 (there could be sidings on Sections 00 and 02 but they are not represented).

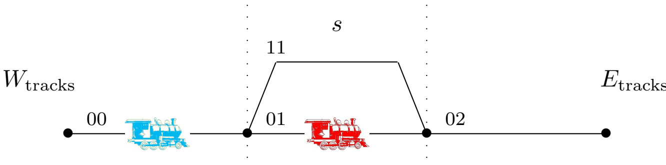


Figure 14: Local configuration just before an unstable event (Section 02 is not necessarily free).

We run the DEADAALG algorithm until we reach a train configuration, say c just before an unstable event occurs. The c train configuration then looks like the one of Figure 14.

Then, we perform all the moves of the DEADAALG algorithm in the E_{tracks} part of the system until the first train (the red or the blue) enters Section 02. At this point, we move the red train in Section 02 and the blue one in Section 01 or 11, according to the section occupied in the current train configuraiton. Then, we perform all the moves in the W_{tracks} part that have been passed.

At this stage, the two train configurations (c and the one under construction) are equivalent, because there is only one possible change. It is an exchange of the blue and red trains, which does not matter as the trains move in the same direction, so can perform exactly the same moves.

So, we can then simulate the next moves of the DEADAALG algorithm, with a possible exchange of the blue and red trains.

We have built a train configuration that does not have an unstable event that the initial did not have, but has one less. Then, the hypothesis is false and the solution has zero such unstable events. □

Lemma 5 *For all systems, if* RESERVATION *succeeds, then its output configuration is valid (i.e., at most one train per section, and each train is on one section) and reachable from the input configuration, following Scheduling Rule 1.*

Proof. Firstly, we show that the output configuration is valid, under the assumption that the input configuration is valid. A reservation of train t can change to requested on a section s only from a pending reservation, when RESERVATION is completed (see Lemma 2). Suppose that there was a requested reservation for train t' on s, at this stage of the reservation process.

If t had reserved s after t' (or if t' is requested because it is its starting section), $\text{STATE}_s(t')$ can not be pending, because t would have been placed ahead of t' in $\mathcal{L}_s$ and its RESERVATION call would have been completed before t', so $\text{STATE}_s(t')$ would still be pending. So, by Lemma 2, $\text{STATE}_s(t')$ was already requested. Then, by Rule 5, t' would have completed a new RESERVATION call from s, and so $\text{STATE}_s(t')$ would have been confirmed when t completed, which is impossible.

Then, t' has reserved s after t, and has completed its reservation process before t. So t was in state pending on s when t' reserved s. As $\text{STATE}_s(t)$ changed from pending to requested on s, s is a siding, by Lemma 2.

So t has launched a RESERVATION call for another train when t reserved on s. The only rule authorizing it (Rule 5) means that there was a requested reservation on s when t reserved, and as t did not complete its reservation, this reservation had state initiating when t' reserved, which is impossible, as t' succeeded its RESERVATION call.

So, in the output configuration, there is one requested reservation per train, and there can not be more than one requested reservation on the same section. Then, this configuration is valid.

We now show that Scheduling Rule 1 induces a schedule from the input configuration to the output configuration.

We sort the successful reservations with respect to the stage they have been accepted, and we make the trains move to the last section of this RESERVATION call. This order is strict as one reservation is made at a time. For example, on Figure 13, the order would be: cyan train to Section 05, yellow train to Section 03, green train to Section 10, blue train to Section 13, purple train to Section 15, cyan train to Section 08, yellow train to Section 05 and red train to Section 03.

It remains to show that the trains can move to the section they have successfully reserved in this order, and that in these sections, when the previous trains have moved, they are placed in first position.

Assume this is not the case. We consider the first train t for which this statement is false. We call τ the stage at which t has successfully reserved the route it is taking.

Suppose there is one train t' on the route. This train had not successfully reserved its route at τ, because otherwise, it would have moved before. Then, we have a contradiction because the only two possible cases are:

- If t' was initially there, it would have successfully reserved its route out before τ by Rule 5, so this case is impossible. It would appear for example if the red train tries to move before the yellow in Figure 13.
- If t' was not initially there, it means that it had been moved, and so validates the assumption. So it was placed in front of t on this section, and there are two cases.
  - Either t was placed behind the successful reservation of t', and then t' has reserved its route out before τ, which is impossible.
  - Or t' reserved after the pending reservation of t, before τ, and has been so ranked in front of t. But there also, t' must have reserved its way out before τ. This case could appear for example if the cyan train moves to Section 05, and then the yellow train to Section 03, then to Section 05, as in Figure 13.

Then, the only possibility is that there is still a train t' placed in front of t on a section. But as t' has successfully reserved its way after τ, it can not be placed before t, because a reservation can not be placed in front of a confirmed reservation.

So we have a contradiction, and the final state is reachable. □

Lemma 6 *If a* RESERVATION *succeeds, then, on the set of spanned rail track sections, in the output train configuration, for each direction, there is a sequence of sections without any train in the opposite direction, going through the latter rail track.*

Proof. Firstly, the reservations can only end on siding sections, and so, on the considered subset of sections, there is no train on sections corresponding to segments.

Then, the only case that could contradict Lemma 6 is when two trains going in the same direction are on sections corresponding to the same siding. Suppose there is such a siding. Let us consider the last reservation that was made over it, call τ the corresponding reservation stage, and see which case of Rule 6 was applied.

If one train was currently trying to reserve a route, no other train reservation can have been filed above the first one at τ, because such a reservation would fail. So, in the final state, this section is empty contrary to our hypothesis.

If the siding was free or occupied by at least one train in the opposite direction, we reserve a section and the other one is either free or occupied by a train in the opposite direction, which is again contrary to our hypothesis.

If the siding was occupied by exactly one train in the same direction, we reserve on the same section so as to leave the other one free, and it is again contrary to our hypothesis.

So, the only case left is the occupation by two trains in the same direction, and we force the two trains to launch a RESERVATION process, so both of them leave the siding, and one section is then free. □

Proof of Theorem 3. We assume that the input configuration c^{IN} is solvable, and that the algorithm has succeeded. Then, by Lemma 5, the output configuration c^{OUT} is reachable, and by Lemma 6, on the subset S' of sections considered by the process, for each direction, there is a route with no trains in opposite direction, see the illustration on Figure 15.

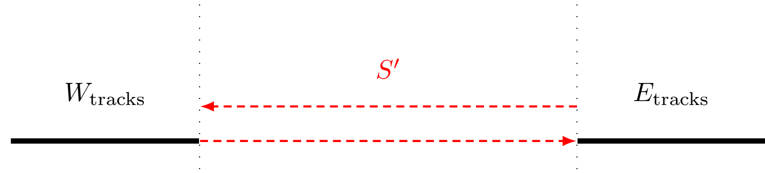


Figure 15: The reservation process has only and entirely considered the subset S' of sections, entirely and only, so W_{tracks} and E_{tracks} are not changed.

Let n^{SIDINGS} be the number of sidings on S', and n^{WEST}, n^{EAST} be the number of Westbound (called W-trains) and Eastbound (called E-trains) trains on S' (these numbers are the same in the initial and final states). We have $n^{\text{WEST}} \leq n^{\text{SIDINGS}}$ and $n^{\text{EAST}} \leq n^{\text{SIDINGS}}$ as the trains are on sidings and two trains going in the same direction cannot be on the same siding.

Thanks to Lemma 4, we know that we have a solution SOL^{IN} for c^{IN} with no train entering a siding occupied by a train going in the same direction. Therefore, we can assume that in SOL^{IN}, all the W-trains have one section of each siding assigned, and the E-trains have the other one, and consequently, when a train enters a siding, it will chose it according to its direction (to simplify the proof, without any loss of generality). The sequence of configurations is denoted by $(c^k)_{k \in K}$, where between two consecutive configurations, exactly one train t_k moves by exactly one section, from section s_k^1 to s_k^2.

From c^{OUT}, we can go to configuration c^{NEXT} where the n^{WEST} W-trains of S' are distributed in such a way that there is one train per siding (one train for the 2 sections of each siding, not one train by siding section) on the westmost n^{WEST} sidings of S', and similarly for the E-trains, on the eastmost sidings, and W_{tracks} and E_{tracks} are unchanged. The choice of the siding section is done by the above selection rule. We add a counter to each train t of S', initiated by the number of trains which are ahead of t in the same direction, in S' (including the section on which each train is), in c^{NEXT} minus this number in c^{IN}. We will show in Lemma 7 that those counters always remain non negative.

Then, we build a solution SOL^{NEXT} for c^{NEXT}. For each step of the solution SOL^{IN}, from SOL^k to SOL^{k+1}, we use the following move process with the current train configuration c^k:

- if the move is internal of W_{tracks} or of E_{tracks}, we perform the same move.
- if the move is going out of S', we perform the same move.
- if the move is going in S', we perform the same move, and initiate the counter of this train at 0.

- if the move is internal of S': if there is a train in the same direction on s_k^1 in c^k with a null counter, we perform the same move. Otherwise, if we have a train in the same direction on s_k^2 in c^k with a nonzero counter, we decrement it, but do not move any train. Otherwise, we do nothing.

An illustration of this process for some train configurations can be found in Figure 16.

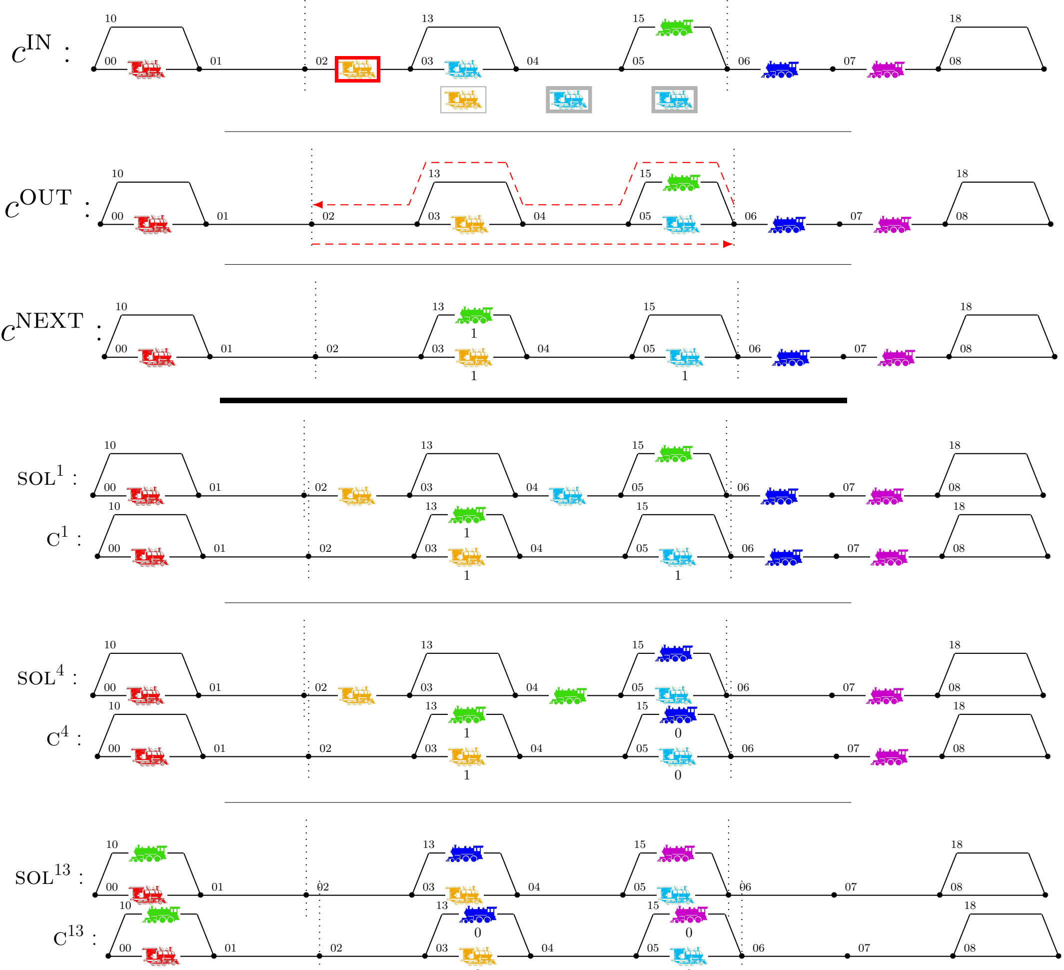


Figure 16: Process of lemma 7: some configurations and counters are represented. At the end, all the counters are equal to 0, consequently the two solutions continue the same moves.

Then, we have to prove that this move process is valid and corresponds to a solution (feasible schedule), so that when "we perform the same move" is executed, there is a train on s_k^1 and none on s_k^2: the correctness of this claim is included in Lemma 7, which follows.

Moreover, at the end, the system is empty. Indeed, it is clear for both W_{tracks} and E_{tracks}, as they are equivalent in the solutions (Lemma 7), and with respect to S', there are as many trains that enter and exit S' in both SOL^{IN} and SOL^{NEXT}, so S' is also empty.

Then, SOL^{NEXT} is a valid solution for c^{NEXT}, so c^{NEXT} and then c^{OUT} are both solvable.

Lemma 7 *For all $k \in K$, the following statements are true.*

1. *The solution is valid until the k-th iteration.*
2. *In c^k and SOL^k, rail sub-networks W_{tracks} and E_{tracks} are equivalent (i.e., there is a W-train in a section in c^k iff there is a W-train in the same section in SOL^k, and similarly for E-trains).*
3. *In S', for each train t with a nonzero counter on section s in c^k, the counter is positive and equal to the number of trains running in the same direction ahead of that section s in S' (including s) in c^k, minus this number in SOL^k. These trains are on sidings, and we never add nor move a train with a nonzero counter.*

   Moreover, if there is a null counter train, then there is a train running in the same direction on the same section in SOL^k.
4. *In S':*
   - *there are as many trains in c^k as in SOL^k.*
   - *if a train t is on a section s in SOL^k, then either there is a train running in the same direction with a null counter on s in c^k, or there is at least one more train running in the same direction strictly ahead of s (according to the direction of t) in c^k than in SOL^k.*

Proof. We prove these statements by induction on k.

For $k = 0$, the solution is valid as there are no move yet, and W_{tracks} and E_{tracks} are equivalent because not changed as not concerned by the process.

In S', in s_k, all the trains that have a non-null counter verify the condition as they are initiated by this value. It cannot be negative as the trains move forward and do not exit S'. If a train has a null counter, this means that the train has not moved between c^{IN} and c^{NEXT}, and so there is the same train in SOL^k. Furthermore, there are as many trains in S' in c^{NEXT} than in c^{IN}.

Consider a train in S' in SOL^k. In c^k, the trains going in the same direction can only have moved forward, so there are at least as many trains going in the same direction strictly ahead in c^k as there are trains going in the same direction strictly ahead in SOL^k. If these numbers are equal, this means that the train has not moved, and so there is the same train at the same section in c^k, and it has been initialized to 0. So the statement is true.

Suppose the statements true for k, and prove them for $k+1$. We consider the different possible options for a train move between SOL^k and SOL^{k+1}. Let t be the train involved in the move in SOL^k.

- If the move is within the rail subnetwork W_{tracks} or E_{tracks}, the move is possible because W_{tracks} and E_{tracks} are equivalent at step k (see the definition in the lemma), and all the statements remain true.
- If the move is exiting S', there was a train in SOL^k in S' at s^1_k, which is the last section of S'. Then, obviously, there cannot be any train ahead of s^1_k in S', so there must be a null-counter train in the same direction on s^1_k in c^k, and s^2_k is empty because W_{tracks} and E_{tracks} are equivalent. So, the move is possible and the solution is still valid. We have removed one train from S' in each solution, so the counters remain valid.
- If the move (train t) is entering S', and the only case that could prevent the move is that s^2_k is free in SOL^k and not in c^k. If s^2_k belongs to a segment, the train t' occupying it has a null counter, so it contradicts Statement 3 (last component of it). Then, it means that train t' is on a siding, and goes in the same direction as t, and could not have a null counter because of Statement 3. So train t' has a nonzero counter, and was then present in c^{NEXT}, which is impossible, because on s^2_k, the counter is initialized to 0, so the move is valid. The train we add does not modify the other counters, we initialize its counter to 0, so the resulting train configuration respects the statements.
- If the move takes place within S':

- if there is a train running in the same direction in s_k^1 in c^k with a null counter, then the only case that could prevent the move is that s_k^2 is empty in SOL^k and occupied by a not-null-counter train in c^k. But this counter would then be negative, as the previous train has a null counter, so it is impossible and the move is valid. We do not modify counters so the statements remain true for $k+1$.
- otherwise, if there is a nonzero counter train in the same direction on s_k^2, the train moves in SOL^{IN}, but none moves in SOL^{NEXT}, and we decrement the counter of this train, which remains non negative. The counter remains consistent as there is one more train in c_{k+1}. If the counter is nonzero, then the number of trains in the same direction and strictly ahead of s_k^2 in c^k is larger than this number in SOL^k, so the statements remain true for $k+1$.
- otherwise, $c^{k+1} = c^k$, and the statements remain true: there cannot be a null-counter train in the same direction on s_k^1, so no null-counter trains of c^k is associated with an empty track in SOL^k; and the values of the counters are unchanged, with the counters remaining valid.

So, by induction, the statements are true. □

□

B.3 Theorem 8

We want here to prove the theorem:

Theorem 8 *If* RESERVATION *fails, then the input train configuration was not solvable.*

To prove Theorem 8, we will use the following lemmas.

From now on, we consider the function MODIF, which forgets Rule 6, and allows the user, at each siding, to choose the section the train should try to reserve. Then, as soon as there is a failure, the MODIF function fails without trying any other choice.

Lemma 9 *If* RESERVATION *fails, all the choices would also fail in* MODIF.

Proof. Suppose that RESERVATION fails, and one MODIF function succeeds. We consider the successful MODIF function M with the latest first choice different from RESERVATION, and the configuration just before the first different choice, and so, the reservations made at this point are exactly the same.

We explore the cases of Rule 6, for the reservation of train t, and show for each case either a contradiction with the results of the functions, or that this different choice is not mandatory, and then we contradict the hypothesis and prove the lemma.

- one train has state initiating: M tries to reserve the same section, and then fails.
- the siding is free or occupied by two trains running in the direction opposite to t's: the two choices are equivalent, so this different choice is not mandatory.
- the siding is occupied by one train running in the direction opposite to t's: RESERVATION would have succeeded immediately.
- the siding is occupied by one train running in the same direction as t: M reserves the free section. By Lemma 4, this choice is not mandatory, there exists a solution that can be found reserving the occupied section.
- the siding is occupied by two trains going in the same direction as t : here, the choice of the reservation does not matter, but we could chose to firstly reserve the train that is on the reserved section, before the other one. Then, as M is successful, this reservation is successful. And then, we are now in the case of one train running in the same direction as t, because a successful reservation is considered as a movement by the process. So this different choice is not mandatory.

- the siding is occupied by two trains going in different directions: M tries to reserve the train going int the direction opposite to t's. This choice is not mandatory, because, by Lemma 4, there is a solution without entering the siding before the train running in the same direction as t has left it. So M would have to launch RESERVATION on the train running in the same direction as t before the current reservation succeeds, and then we come down to the choice of RESERVATION.

□

We say that two trains are equivalent if they are on the same siding, and go in the same direction.

For the following lemma, we suppose that the reservation process has not yet created successful reservations, and that there was no successful reservations before (i.e., these trains have already moved).

Lemma 10 *If a reservation process fails (either* RESERVATION *or* MODIF*), there is no solution with each train passing through all the sections it has reserved, and satisfying the rule of Lemma 4.*

Proof. If a reservation process fails, then we have a reservation circular wait among the trains $t_1, t_2 ... t_n, t_{n+1}$, $t_1 = t_{n+1}$ as the reservation of train t_i directly implies the reservation of train t_{i+1}.

At each extremity of the rail sub-network considered by the process, there is one train going towards the interior of this sub-network, either on a section or on a siding.

Strictly inside this sub-network, all the trains that are involved in the process, are on sidings and all the sidings are occupied by two trains running in opposite directions. Indeed, a reservation process cannot go West and East through a section occupied by a train, so the trains are not on segments, and no siding is occupied by two trains running in the same direction. Then, if a siding has at least one section free, the process cannot go through it West and East, it would stop in at least one direction, when the process requests a free section.

Assume now that there exists a solution satisfying Lemma 10, and that there is an interior siding. We consider the first train to move in this part of the track.

- if it is at an extremity: this last train is equivalent to the train involved at this section, so the circular wait is not changed and we can proceed with the next move, and the sub-network is reduced by one section (there is a finite number of moves in that case, so we will have the other case, sooner or later).
- otherwise, a train leaves an interior siding to move in a segment: if this train launches a reservation process without changing the choices, we would have a similar circular wait, but not implying the trains behind it (at least 2), and with at least one siding less.

Then, we iterate this method until we have no interior siding: two trains running in different directions want to exchange their role without being able to meet, so there is no solution and we have a contradiction.

□

Proof of Theorem 8. We suppose that a reservation process fails.

If this reservation process implies successful reservations, we can firstly do all these reservations. By Theorem 3, we do not create deadlocks this way. Then, we apply the reservation process on the same train, and we obtain the same result, because the reservation process considers successful reservations as movements (confirmed reservations are not considered, and a successful reservation means a move of the requested reservation of the trains).

Then, we apply the result of Lemma 9: for all choices, the MODIF functions fail. Then, by Lemma 10, for all choices, there is no solution with the trains passing through the sections they have chosen.

To conclude, there is no solution for this configuration, so there is no solution for the input train configuration. □

B.4 Proof of Theorem 1

Complexity component of the Proof. We cannot call two times RESERVATION_SEC(t, s) on the same couple (t, s), so it cannot be called more than $|S|.|T|$ times, and the Occupying function cannot be called more than $O(|S|.|T|)$.

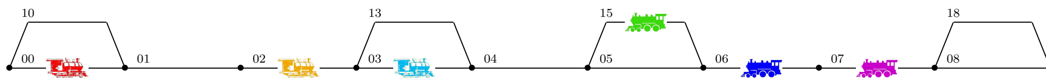
If we implement $\mathcal{L}_s$ by double linked lists and keep pointers to the last train in state confirmed in each list, we get a total complexity of $O(|S|.|T|)$. Indeed, the pointer can be updated in constant amortized complexity, as the position of the last confirmed reservation can only increase. Then, the insertions of Rule 3 can be done in constant time. □

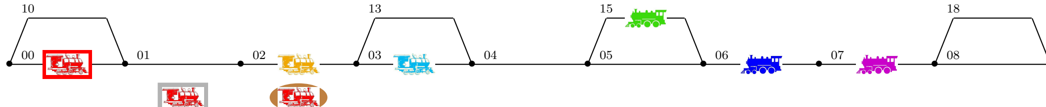
Correctness component of the Proof. If the departure plan is solvable, the first call to RESERVATION succeeds (Theorem 8) and leads to a solvable configuration (Theorem 3). Then, by recurrence, all the calls succeed (Theorem 8) and lead to solvable configurations (Theorem 3), so the trains have all reached their destination in the final configuration (only possibility of stop for DEADAALG), and Rule 1 exhibits a scheduling free of any deadlock (Theorem 3).

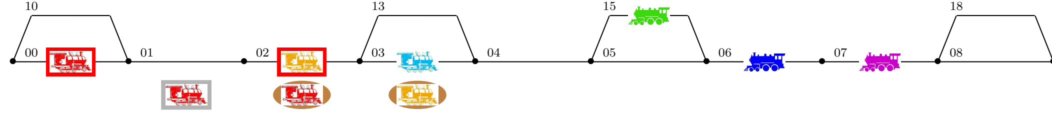
If the departure plan is not solvable, when DEADAALG ends, trains must remain in the system, so it has failed, which indeed means that there is no solution. □

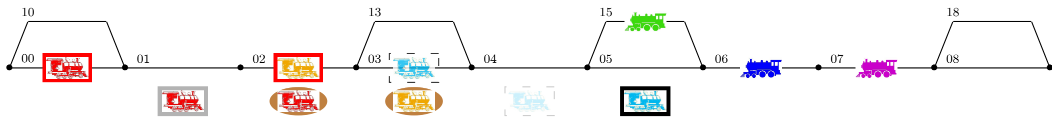
C Illustration of the DEADAALG Algorithm

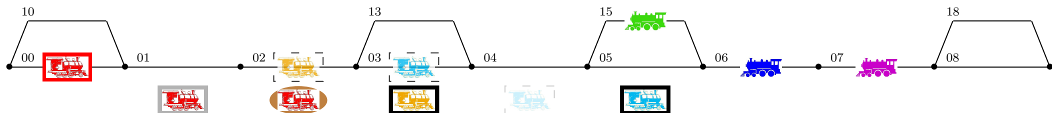
.

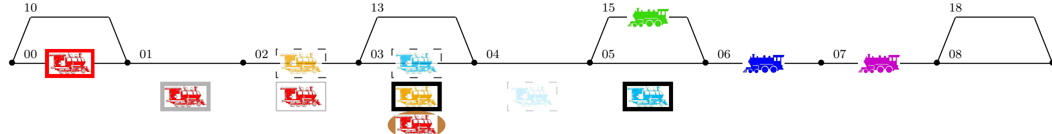


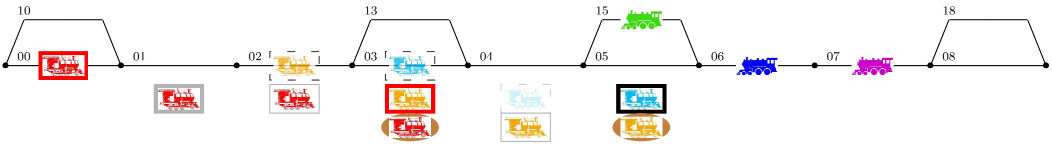


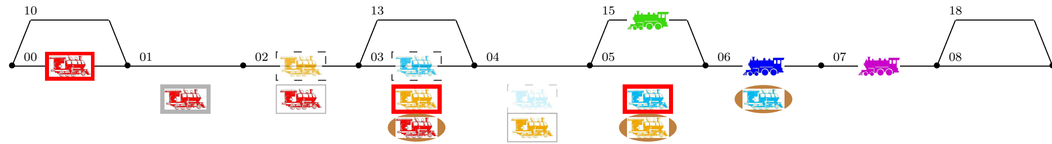


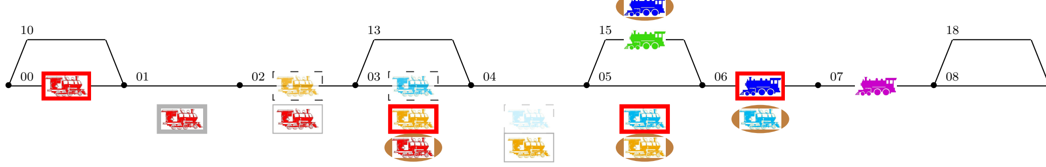


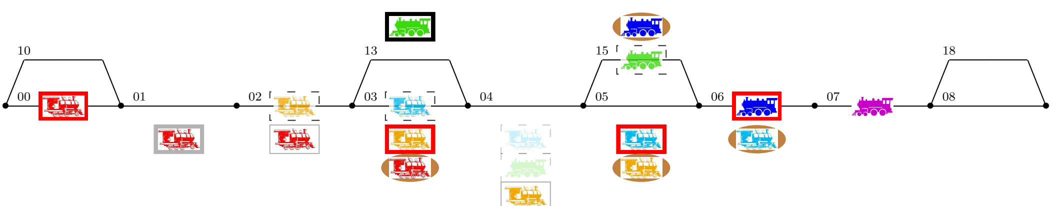


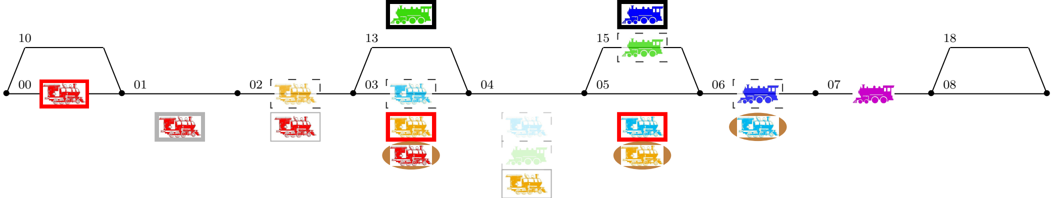

10
13
15
18
00
01
02
03
04
05
06
07
08

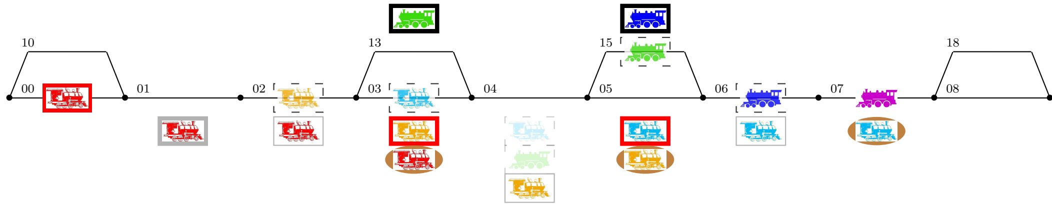

10
13
15
18
00
01
02
03
04
05
06
07
08

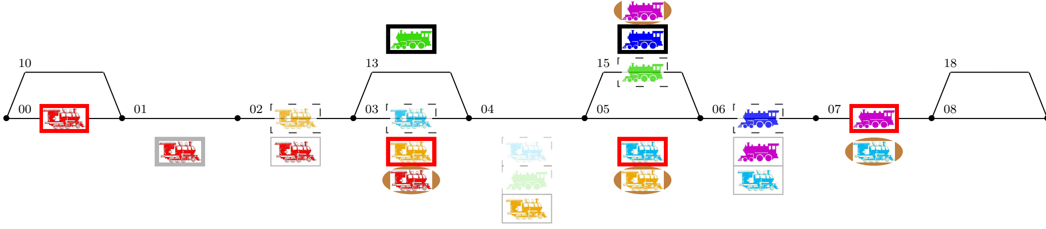

10
13
15
18
00
01
02
03
04
05
06
07
08

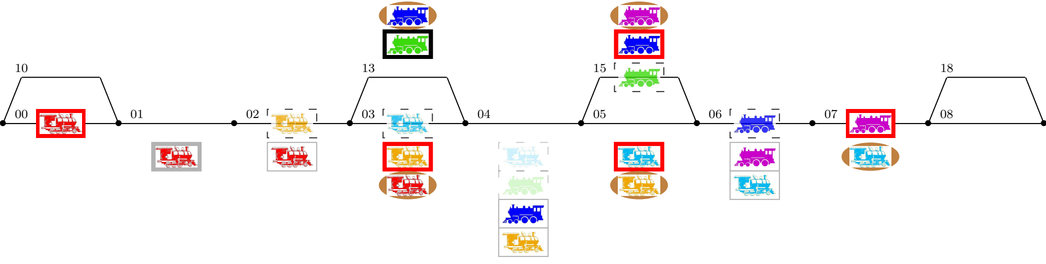

10
13
15
18
00
01
02
03
04
05
06
07
08

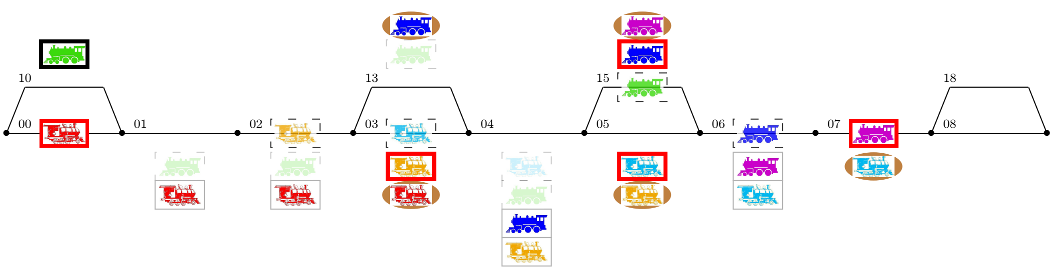

10
13
15
18
00
01
02
03
04
05
06
07
08

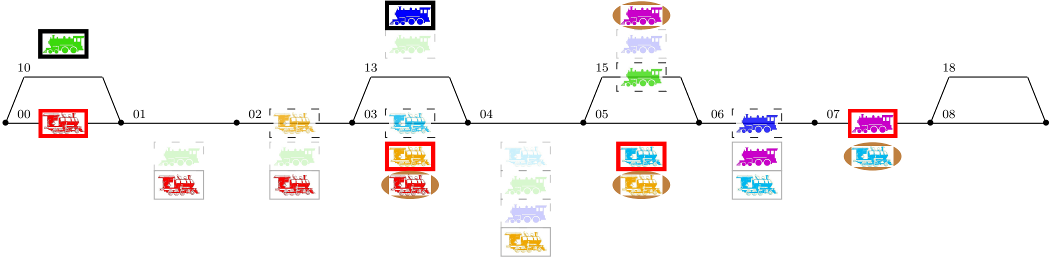

10
13
15
18
00
01
02
03
04
05
06
07
08

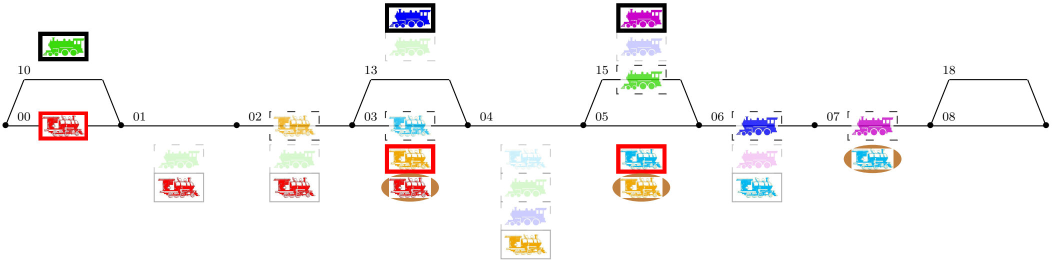

10
13
15
18
00
01
02
03
04
05
06
07
08

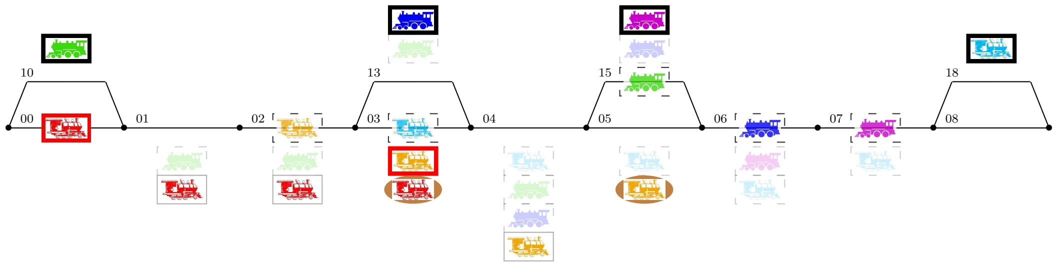

10
13
15
18
00
01
02
03
04
05
06
07
08

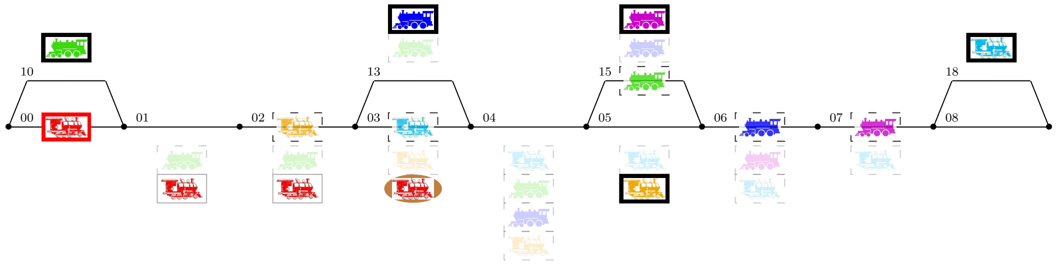

10
13
15
18
00
01
02
03
04
05
06
07
08

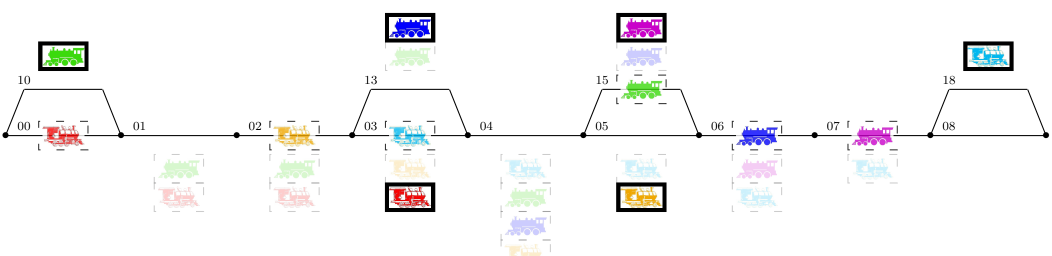

10
13
15
18
00
01
02
03
04
05
06
07
08

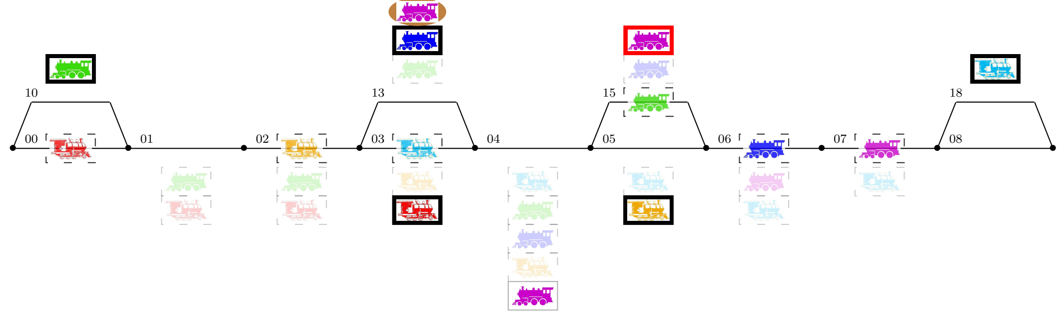

10
13
15
18
00
01
02
03
04
05
06
07
08

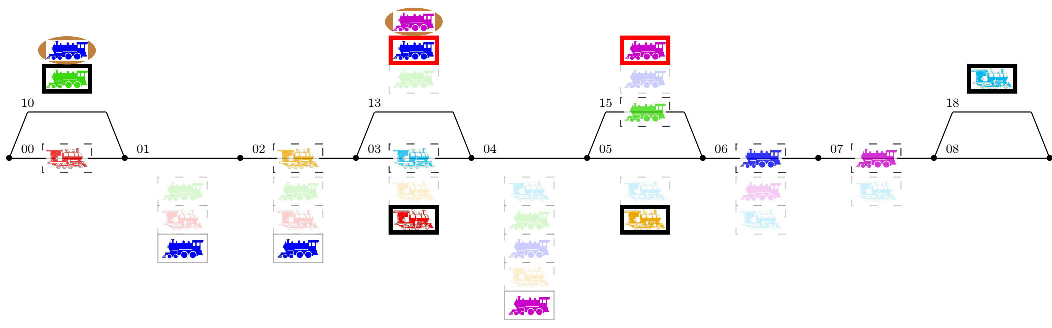

10
13
15
18
00
01
02
03
04
05
06
07
08

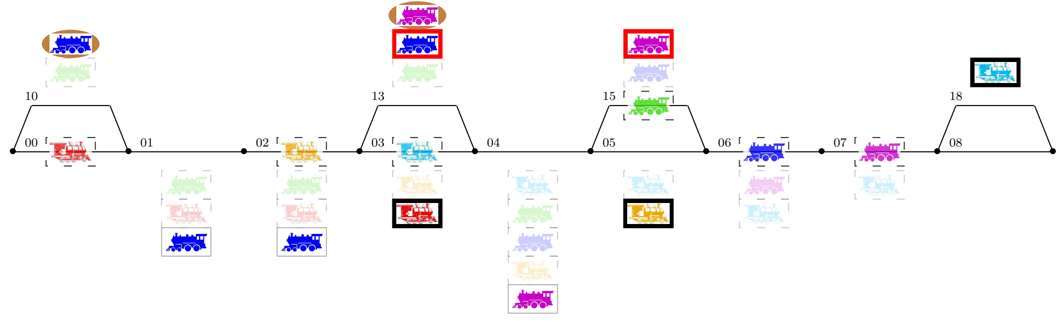

10
13
15
18
00
01
02
03
04
05
06
07
08

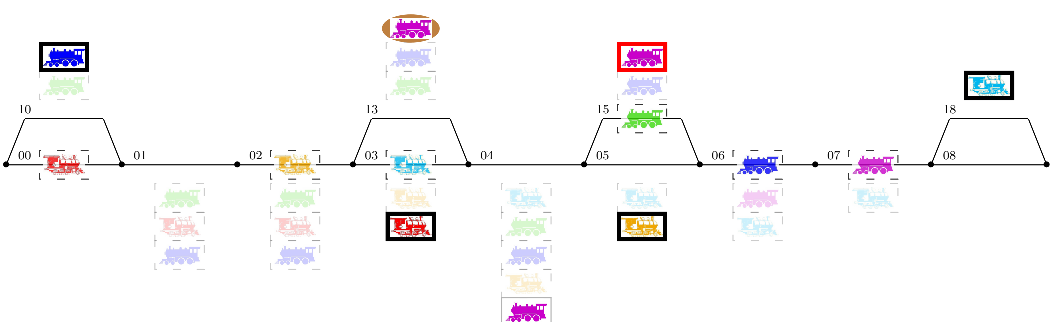

10
13
15
18
00
01
02
03
04
05
06
07
08

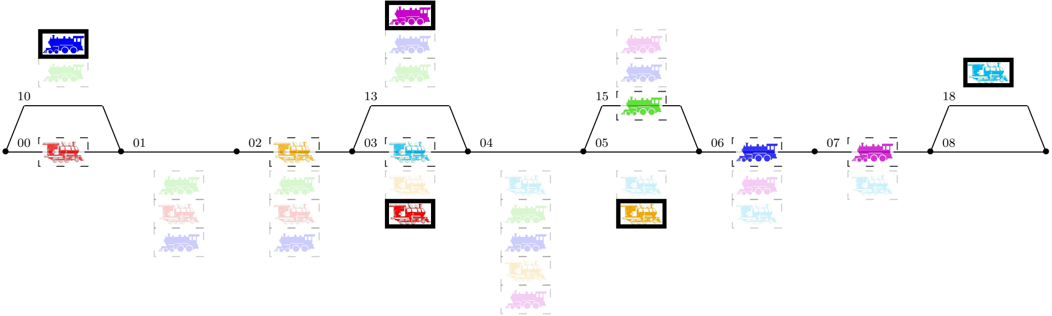

10
13
15
18
00
01
02
03
04
05
06
07
08

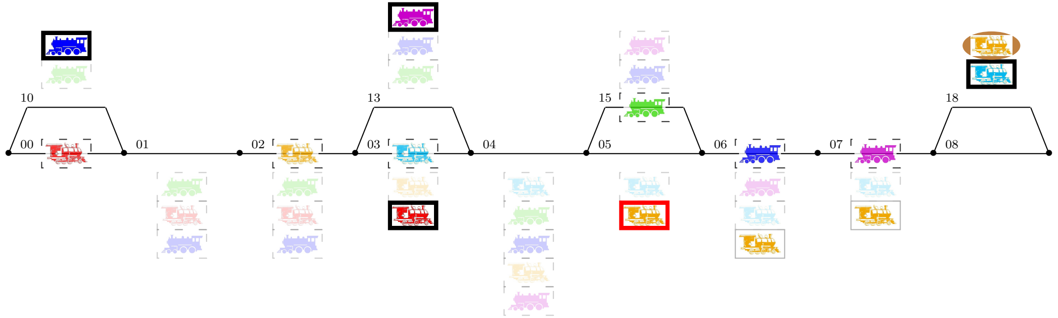

10
13
15
18
00
01
02
03
04
05
06
07
08

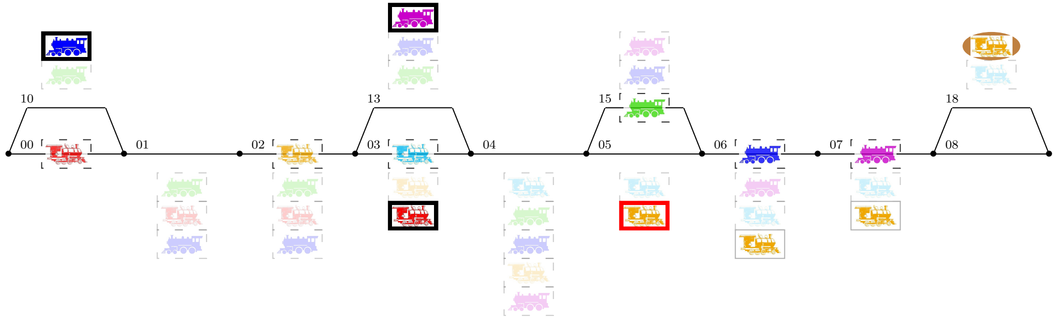

10
13
15
18
00
01
02
03
04
05
06
07
08

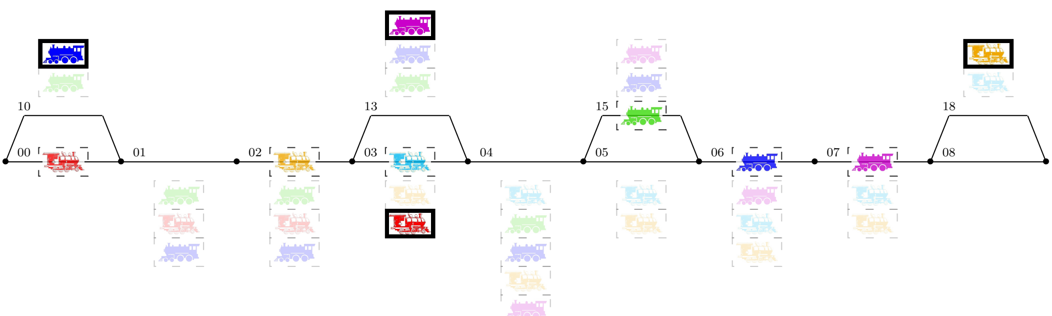

10
13
15
18
00
01
02
03
04
05
06
07
08

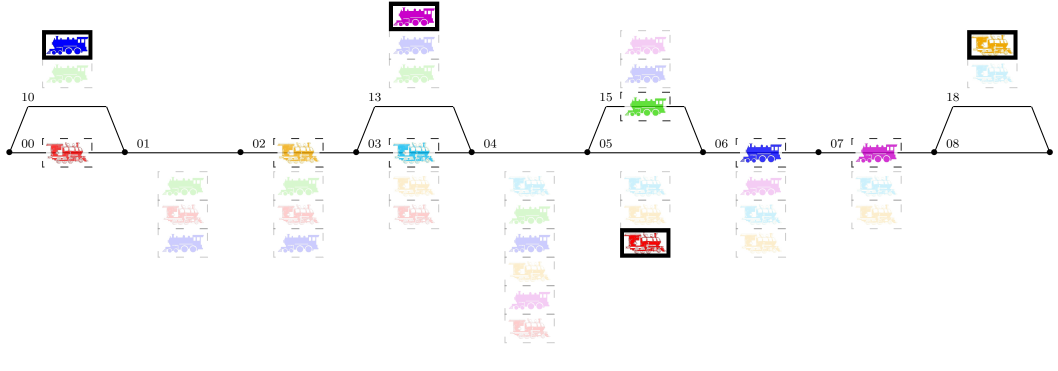

10
13
15
18
00
01
02
03
04
05
06
07
08

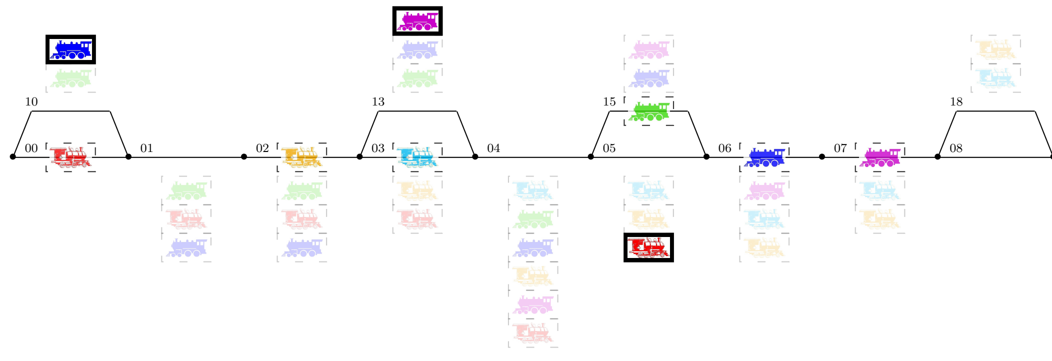

10
13
15
18
00
01
02
03
04
05
06
07
08

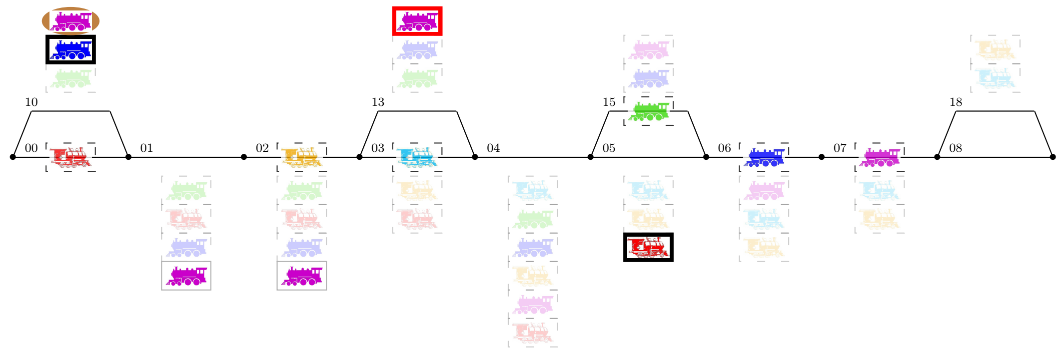

10
13
15
18
00
01
02
03
04
05
06
07
08

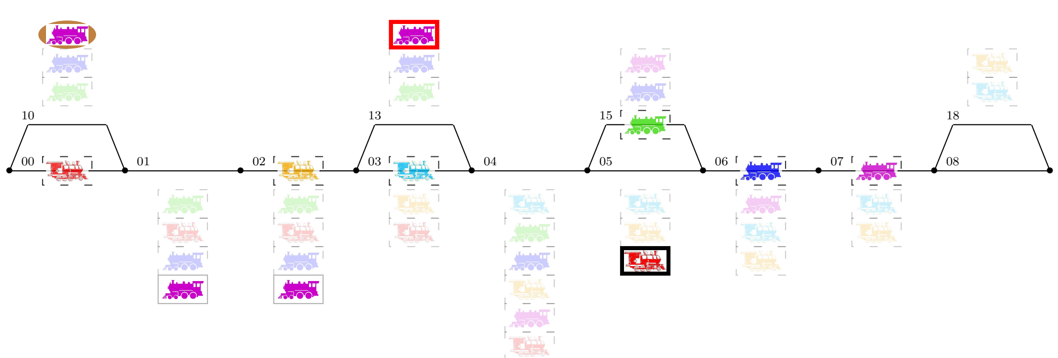

10
13
15
18
00
01
02
03
04
05
06
07
08

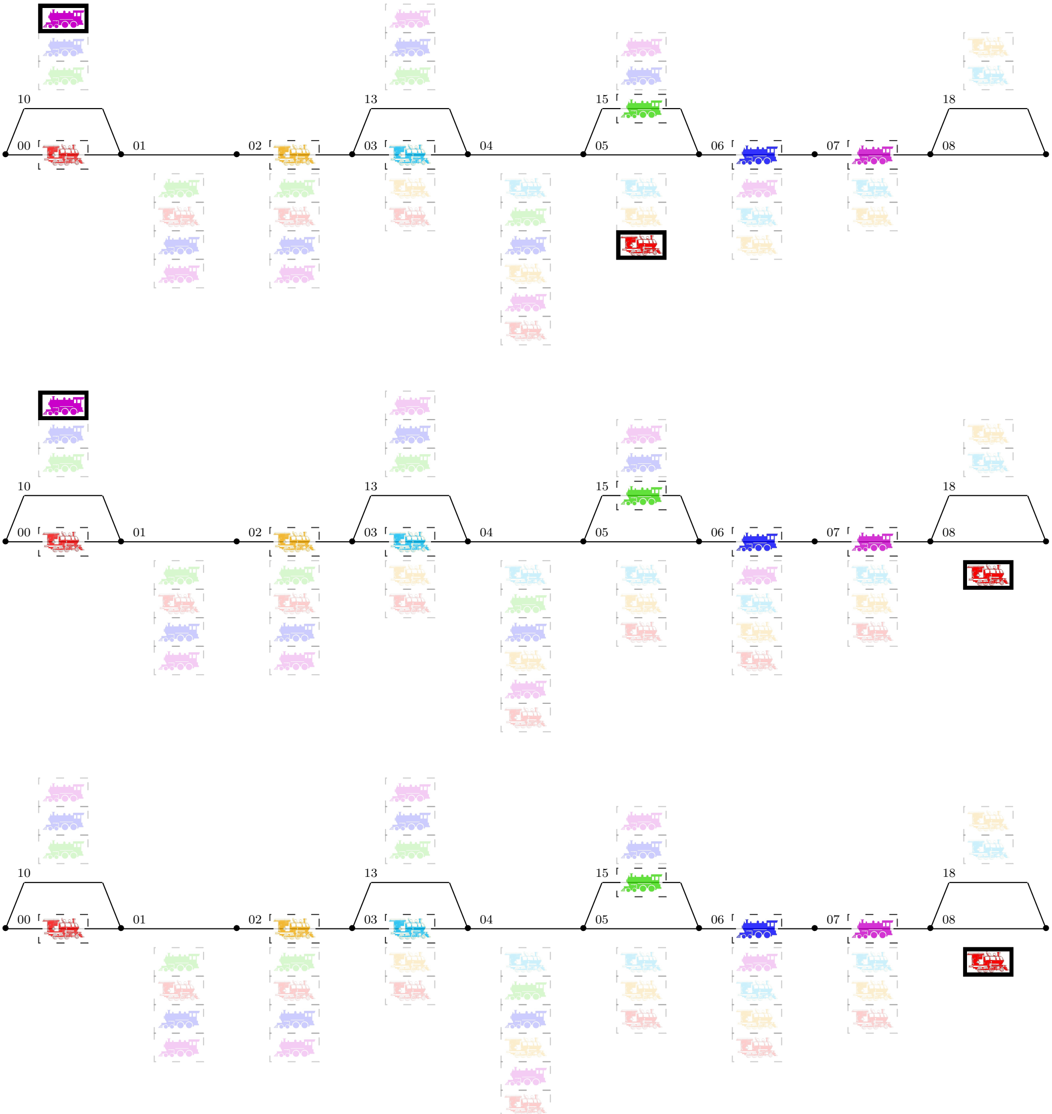

10
13
15
18
00
01
02
03
04
05
06
07
08


10
13
15
18
00
01
02
03
04
05
06
07
08


10
13
15
18
00
01
02
03
04
05
06
07
08


10
13
15
18
00
01
02
03
04
05
06
07
08


10
13
15
18
00
01
02
03
04
05
06
07
08


10
00
01
02
13
03
04
15
05
06
07
18
08

References

[1] F. Belik. Deadlock avoidance with a modified banker's algorithm. *BIT*, 27:290–305, 1987.

[2] F. Belik. An efficient deadlock avoidance technique. *IEEE Transactions on Computers*, 39:882–888, July 1990.

[3] Y. Cui. *Simulation-Based Hybrid Model for a Partially- Automatic Dispatching of Railway Operation*. PhD thesis, University of Stuttgart, Germany, 2010.

[4] E.W. Dijkstra. Cooperating sequential processes. Technical Report EWD-113, Technical University, Eindhoven, The Netherlands, 1965.

[5] P. Grube, F. Nú nez, and A. Cipriano. An event-driven simulator for multi-line metro systems and its application to santiago de chile metropolitan rail network. *Simulation Modelling Practice and Theory*, 19:393–405, January 2011.

[6] P. Ireland, R. Case, J. Fallis, C. Van Dyke, J. Kuehn, and M. Meketon. The Canadian Pacific Railway transforms operations by using models to develop its operating plans. *Interfaces*, 34(1):5–14, 2004.

[7] B. Jaumard, H.T. Le, H. Tian, A. Akgunduz, and P. Finnie. An enhanced optimization model for scheduling freight trains. In *Proceedings of Joint Rail Conference (JRC)*, pages 1–10, 2013.

[8] F. Li, Z. Gao, K. Li, and L. Yang. Efficient scheduling of railway traffic based on global information of train. *Transportation Research, Part B*, 42:1008–1030, 2008.

[9] J. Pachl. Avoiding deadlocks in synchronous railway simulations. In *2nd International Seminar on Railway Operations Modelling and Analysis*, Hannover, Germany, 2007.

[10] J. Pachl. Deadlock avoidance in railroad operations simulations. In *90th Annual Meeting of the Transportation Research Board*, Washington DC, USA, 2011.

[11] I. Sahin. Railway traffic control and train scheduling based on inter-train conflict management. *Transportation Research Part B: Methodological*, 33:511–534, September 1999.

[12] S. Toueg and K. Steiglitz. Some complexity results in the design of deadlock-free packet switching networks. *SIAM Journal on Computing*, 10:702–712, 1981.