Proceedings of the 2014 Joint Rail Conference
JRC2014
April 2-4, 2014, Colorado Springs, CO, USA

JRC2014-3864

DEADLOCK AVOIDANCE AND DETECTION IN RAILWAY SIMULATION SYSTEMS

Bertrand Simon
Ecole Normale Suprieure, Lyon, France
Email: bertrand.simon@ens-lyon.fr

Brigitte Jaumard and Thai Hoa Le
Computer Science and Software Engineering
Concordia University
Montreal (Qc) Canada H3G 1M8
Email: bjaumard@cse.concordia.ca

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 exact, i.e., either detects an unavoidable deadlock resulting from the input data or provide a train scheduling thanks to the scheduling algorithm, called SIMTRAS, which is free of deadlocks. Moreover, we show that the SIMTRAS algorithm is a polynomial time algorithm with an $O(|S| \times |T|^2 \log |T|)$ time complexity, where T is the set of trains and S is the set of sections in the railway topology. Numerical 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.

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 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 devices in order to detect and avoid deadlocks, and provide meaningful results on a network operated under conditions close to its full network capacity. Indeed, as soon as a deadlock is encountered, any simulation will stop, and very often it forces to enter manually some modifications (e.g., train t will move before train t' on segment rail s) in the data set before rerunning the simulation. 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 have been studied not only for train systems, but for different types of resource systems, e.g., [1]. 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 [2] is to use a myopic look ahead test to avoid deadlocks, i.e., allow 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 Bankers and the Bankers like algorithms, as it is based on a track section reservation mechanism. In addition, the DEADAALG algorithm has a $O(|S| \times |T|)$ 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 train scheduling algorithms, in the context of the design of a simulation tool. The paper is organized as follows. We next review the most recent results on deadlock avoidance in railway systems. We then discuss deadlock avoidance and detection, as well as a basic train scheduling in the context of train scheduling simulation. In the following section, we detail the newly proposed deadlock avoidance algorithm, with the proof of its correctness and complexity. It is followed by the description of an efficient scheduling algorithm, the SIMTRAS algorithm, which is deduced from the DEADAALG algorithm and has a $O(|S| \times |T|^2 \log |T|)$ time complexity. Numerical results are presented on several data set instances in order to evaluate the performance of the DEADAALG and SIMTRAS algorithms and their performance comparison with an exact optimization algorithm for train scheduling. Conclusions are drawn in the last section.

LITERATURE REVIEW

While there have 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 Bankers algorithm [3], as well as its modifications, see, e.g., [4, 5] 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 [6], Pachl proposed a new deadlock avoidance method, called Dynamic Route Reservation (DRR). The key idea is to consider that when a train requests to leave the section it is traveling 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 DRR 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 DRR algorithm, which can lead to a false diagnosis of unavoidable deadlocks.

Later, Pachl [7] 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 behavior of the algorithm is not defined. Furthermore, a reservation can be confirmed while later it may lead 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 Rule 6 in our proposed reservation process), however, rules in the Pachls algorithm authorize it.

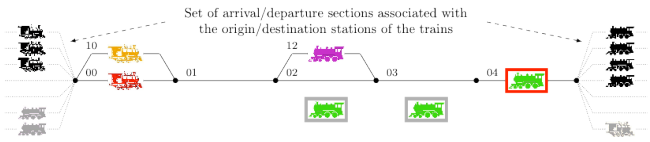


FIGURE 1. An illustration of the drawback of Pachls method [7]

In this paper, we revisit the algorithm of Pachl [7] and completely redesign the reservation mechanism, using 4 reservation states in order to obtain an accurate deadlock avoidance algorithm. In addition, we modified several of the rules of Pachls algorithm [7], making use of the four reservation states in order to avoid deadlocks whenever possible, and making sure that the resulting algorithm encounters a deadlock only if there is no alternate way to avoid it (we call it unavoidable deadlocks). In summary, our contribution is, on the one 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.

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 that are separated by sidings or stations, and each segment is divided into a set of sections. Tracks are used by trains traveling in both directions, and trains can meet and pass at stations or sidings. Sidings allow two trains to pass one another and are the most common method used to expand rail network capacity. 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 is described by a set of Eastbound and Westbound trains, denoted by T and indexed by t, with $\text{DEPART}(s,t)$ being the departure time of train t at the origin of its 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 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.

Train scheduling simulators divide into two categories: event-driven (e.g., [8]) and time driven (e.g., [9]) 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 impose some path scheduling 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.

A NEW DEADLOCK AVOIDANCE/DETECTION ALGORITHM

We propose an original $O(|S| \times |T|)$ 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 the SIMTRAS scheduling algorithm), otherwise, an unavoidable deadlock has been identified. The DEADAALG algorithm is an exact algorithm, which is free of arbitrary selections that leads the algorithm of Pachl [7] to sometimes reach wrong conclusions.

Reservation Process

In each section s, we define an ordered reservation $\text{LIST\_RESERV}(s)$. 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, $\text{LIST\_RESERV}(s)$ 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:

$$\text{STATE\_RES}(s,t) \in \{\text{pending}, \text{requested}, \text{initiating}, \text{confirmed}\}.$$

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 $\text{LIST\_RESERV}(s)$ 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., $t = \text{occupying}\ (s)$, if t has an initiating or a requested reservation for s. If no train has such a reservation, then $\text{occupying}(s)$ returns 0.

The reservation process has to obey the following rules:

Rule 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.

Rule 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.

Rule 3. If a reservation is requested on a section that 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.

Rule 4. If a reservation is placed behind an initiating reservation, the reservation fails: it means there is a circular wait, i.e., a deadlock.

Rule 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.

Rule 6. If there is a reservation request of train t for a siding

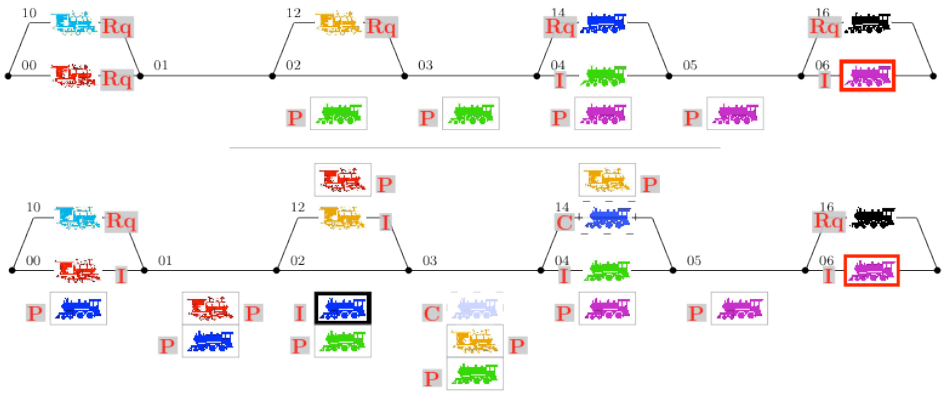


FIGURE 2. An illustration of the DEADAALG algorithm

section, proceed as follows. Let s_a and s_b be the two siding sections.

Rule 6a. There is an initiating reservation on siding section s_a: reserve on s_b.

Rule 6b. The siding is occupied by two trains running in the same direction as ts: the two sections are equivalent, choose, e.g., 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.

Rule 6c. The siding is occupied by two trains running in the direction opposite to that of t: the two sections are equivalent, reserve, e.g., s_a.

Rule 6d. The siding is occupied by a train opposite to that of 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.

Rule 6e. The siding is occupied by one train in the direction opposite to that of t, on s_b: reservation is made on the free siding section, s_a.

Rule 6f. The siding is occupied by one train in the same direction as t, on s_a: reservation is made on the occupied section, s_a.

Rule 6g. The siding is not occupied: proceed with a reservation on any of the two, e.g., s_a.

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 re-launched, 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, LIST_RESERV(s) and LIST_LOCAL(s) that is local to RESERVATION (t). It calls two functions, RESERVATION (t) and RESERVATION _SEC(s,t) that take care of the reservation of sections for t until either DEADAALG ends successfully or fails, and of the reservation of t on section s, respectively. Any fail of RESERVATION _SEC(s,t) entails a fail for RESERVATION (t).

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.

LIST_RESERV$(s) \leftarrow \emptyset$ for all $s \in S$
STATE$(s,t) \leftarrow \emptyset$ for all $(s,t) \in S \times T$
$s \leftarrow$ section on which t is
STATE$(s,t) \leftarrow$ requested for all $t \in T$
► There cannot be two trains with a requested state on the same section, i.e., two trains cannot be on the same section

While no deadlock has been detected or one train remains in the system **do**
Select a train t that has not reached its destination
RESERVATION (t)

Function RESERVATION (t)
Require: For all s,t: STATE(s,t), LIST_RESERV(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 STATE$(s,t) =$ requested
$s^{\text{init}} \leftarrow s$
STATE$(s,t) \leftarrow$ initiating
next_sec(s): returns the section after s on the route of t
LIST_LOCAL$(t) \leftarrow \{s\}$
$s \leftarrow$ next_sec(s)
While s is not a siding track **do** ► Application of Rule 1
    RESERVATION _SEC(s,t)
    LIST_LOCAL$(t) \leftarrow$ LIST_LOCAL$(t) \cap \{s\}$
    $s \leftarrow$ next_sec(s)
if the endpoint of s is the final destination of t then
    STATE$(s',t) \leftarrow$ confirmed for all $s' \in$ LIST_LOCAL(t)
    Terminate function RESERVATION (t)

► Let s be a siding track, and s1 and s2 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(s^{selected},t)
Case 2: Apply any of the other rules with s selected $\rightsquigarrow$ RESERVATION _SEC(s^{selected},t)

STATE$(s,t) \leftarrow$ confirmed for all $s \in$ LIST_LOCAL(t)
STATE$(s^{\text{selected}},t) \leftarrow$ requested

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 a letter next to each train, on the left 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.

Function RESERVATION _SEC(s,t)
Require: Train t ; Section s and its reservation list LIST_RESERV(s) ;
For all $(s,t) \in S \times T$: STATE(s,t) ; For all $s \in S$: LIST_RESERV(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 LIST_RESERV(s)
STATE$(s,t) \leftarrow$ pending
► all the reservations in LIST_RESERV(s) placed after i are pending
If $t' \neq \emptyset$ and STATE$(s,t') =$ initiating then FAIL ► Rule 4
else if there is no pending reservation in LIST_RESERV(s) then ► Rule 2
Add t to the end of LIST_RESERV(s)
If $t' \neq \emptyset$ then RESERVATION (t') ► Rule 5
else Insert t in position i of LIST_RESERV(s) ► Rule 3

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 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.

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.

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 [10] for the detailed proof.

Theorem 1. The DEADAALG algorithm is finite. With an $O(|S| \times |T|)$ 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 MODIFor 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.

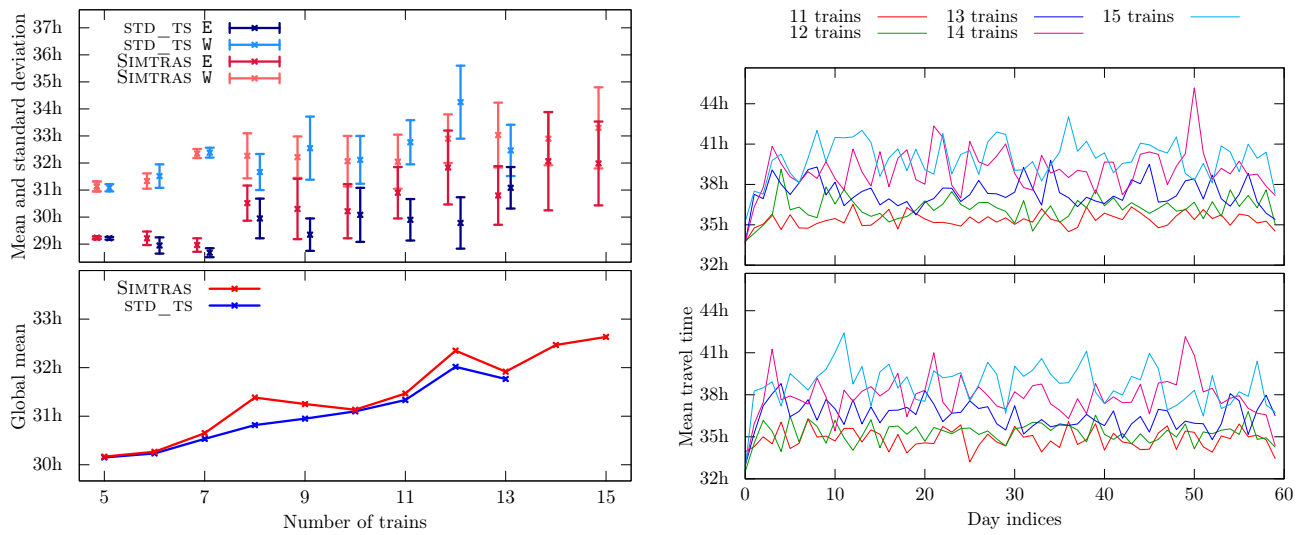
Once these demonstration steps are completed, we can conclude that if RESERVATION fails, all the MODIFfunction calls fail, and we can come down to the case of Step 5 (using the previous result) to show that all choices lead to a deadlock, so the input train configuration was not solvable. This way, we 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(s,t) on the same pair (s,t), so it cannot be called more than $|S| \times |T|$ times. If we implement LIST_RESERV(s) using double linked lists and keep pointers to the last train in the confirmed state in each list, we get a complexity of $O(|S| \times |T|)$.

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.

TRAIN SCHEDULING SIMULATION WITH DEADLOCK AVOIDANCE/DETECTION

We explained 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. However, no speed value, and no distances are taken in account in the DEADAALG algorithm: it worlds as a board game algorithm, yes/no the train can move on the next section. We now discuss how to improve it with respect to the minimization of the average travel times of the trains. The resulting algorithm is called SIMTRAS.

While applying SIMTRAS, the departure time of train t on section s, denoted by DEPART(s,t), can be computed in polynomial time, using the section average travel times (those data were provided by CPR Canadian Pacific Railway in our experiments). DEPART(s,t) is computed in order to prevent unnecessary stops on a segment, by forcing trains to wait on the sidings if they cannot 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 DEPART(s,t) on the s section it is requesting. This way, the distribution of the requested sections at each step is the closest to a snapshot of the future scheduling, and it behaves as a greedy algorithm: at each



(a) One day horizon

(b) 60 day horizon

FIGURE 3. Average travel times of the train scheduling output by the SIMTRAS and SDT_TS algorithms.

step, the first train that reached a segment has the highest priority to travel 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 DEPART(s,t), we use the minimal tardiness. 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 $O(\log|T|)$ 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 reservation lists. The drawback of this modification is to increase the time complexity of the algorithm, because we have to scan a non-constant part of the LIST_RESERVto find the minimum possible time. A basic implementation leads to a $O(|S|\times|T|^2\log|T|)$ complexity. In order to process segment data with average travel times that vary and depend on the direction (East vs. West, going up vs. going down, or empty vs. loaded cars), 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.

NUMERICAL EXPERIMENTS

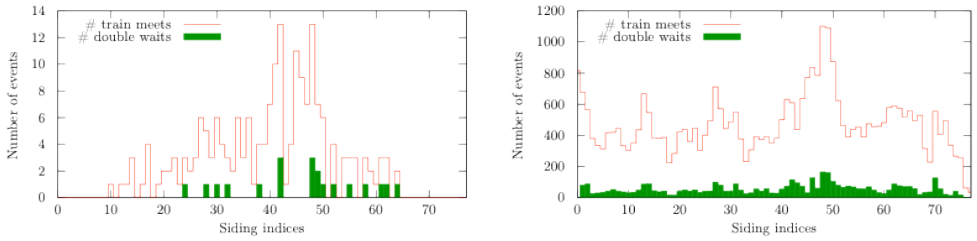
All the algorithms and functions described in the previous section were implemented in C++. We evaluated the performance of the DEADAALG and SIMTRAS algorithms on the Canadian Pacific Railway (CPR) network between Calgary and Vancouver, i.e., the busiest corridor of the CPR network. It is a single track 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 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 $24(1-1/|T|)$ 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.

In Figure 3, we plotted the average travel times of the overall daily fleet of trains obtained with the SIMTRAS algorithm, for different numbers 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

FIGURE 4. A double wait that leads to deadlock avoidance



(a) 14 trains, one day

(b) 14 trains, 60 days

FIGURE 5. Siding usage

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 Figure 3(a), 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.* [11], see the blue (darker) intervals. As for the SIMTRAS algorithm, the blue intervals are centered. 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 33(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 do 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 Figure 3(b), 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 5, we investigate the siding usage firstly over a one day period in Figure 5(a) and then over a 60 day period in Figure 5(b), in addition to the advantage of double wait as illustrated in Figure 4.

While, in practice, trains are usually not allowed to wait on a main track, in a double wait scenario, we allow trains to wait on a main track. We illustrate in Figure 4 the advantage of doing so in order to decrease the number of deadlocks. Indeed, if the blue train waits on the main track, then the green train (extreme right) can move on the siding track, and then waits. The blue train can then move Eastbond, as well as the red train (extreme left). Once the red train passes the siding, the green train can move again Westbound. However, if no train is allowed to wait on the main track, then the situation represented in Figure 4 corresponds to a deadlock.

In Figure 5, 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.

CONCLUSIONS

We have proposed a first exact $O(|S| \times |T|^2 \log |T|)$ and highly scalable deadlock detection and avoidance DEADAALG algorithm for train scheduling. In addition, the SIMTRAS algorithm favorably competes with the SDT_TS exact algorithm of [6] 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 and the SIMTRAS algorithms to sidings with more than 2 alternate tracks.

ACKNOWLEDGMENT

B. Jaumard was supported by NSERC (Natural Sciences and Engineering Research Council of Canada) and by a Concordia University Research Chair (Tier I). .

REFERENCES

[1] Toueg, S., and Steiglitz, K., 1981. "Some complexity results in the design of deadlock-free packet switching networks". *SIAM Journal on Computing,* ***10(4)***, pp. 702–712.

[2] Sahin, I., 1999. "Railway traffic control and train scheduling based on inter-train conflict management". *Transportation Research Part B: Methodological,* ***33***, p. 511534.

[3] Dijkstra, E., 1965. Cooperating sequential processes. Tech. Rep. EWD-113, Technical University, Eindhoven, The Netherlands.

[4] Belik, F., 1987. "Deadlock avoidance with a modified bankers algorithm". *BIT,* ***27***, p. 290305.

[5] Belik, F., 1990. "An efficient deadlock avoidance technique". *IEEE Transactions on Computers,* ***39***, pp. 882–888.

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

[7] Pachl, J., 2011. "Deadlock avoidance in railroad operations simulations". In 90th Annual Meeting of the Transportation Research Board.

[8] Grube, P., nez, F. N., and Cipriano, A., 2011. "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***, January, p. 393405.

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

[10] Simon, B., Jaumard, B., and Le, T., 2013. Deadlock avoidance and detection in railway simulation systems. Cahiers du GERAD G-2013-43, GERAD, Montreal (QC) Canada.

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