Deadlock Avoidance and Detection in Railway Simulation Systems

Bertrand Simon, Brigitte Jaumard, and Thai Hoa Le

Avoiding or preventing deadlocks in simulation tools for train scheduling remains a critical issue, especially when combined with the objective of minimization (e.g., the travel times of the trains). The deadlock avoidance and detection problem is revisited, and a new deadlock avoidance algorithm, called DEADAALG, is proposed based on a resource reservation mechanism. The DEADAALG algorithm is proved to be exact; that is, it either detects an unavoidable deadlock resulting from the input data or provides train scheduling free of deadlocks with the scheduling algorithm SIMTRAS. Moreover, it is shown that SIMTRAS is a polynomial time algorithm with an $O(|S|\cdot|T|^2 \log |T|)$ 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 Canada's Vancouver–Calgary single-track corridor of Canadian Pacific Railway Limited. Then it is shown that SIMTRAS is efficient and provides schedules of a quality that is comparable with that of an exact optimization algorithm in tens of seconds for up to 30 trains/day over a planning period of 60 days.

Although railway companies are still using controllers for 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 travel times (freight trains) or 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 some modifications to be entered manually (e.g., train t will move before train t on segment rail s) in the data set before the simulation is rerun.

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 also for different types of resource systems (*1*). Although it is now a well-solved problem in the context of resources and processes where preemption is possible, it remains a poorly solved problem in the context of trains where train preemption 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 is to use a myopic look-ahead test to avoid deadlocks, that is, to allow a train to move on the next segment if there will be no deadlock in the next two or three segments (*2*). With medium or high train densities, such a myopic vision is not enough to avoid deadlocks.

An original DEADAALG algorithm is proposed for deadlock detection and avoidance that significantly improves on the classical banker's and the banker's-like algorithms since it is based on a track section reservation mechanism. In addition, the DEADAALG algorithm has an $O(|S|\cdot|T|)$ 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 most recent results on deadlock avoidance in railway systems are reviewed next. Then deadlock avoidance and detection are discussed, as well as basic train scheduling in the context of train-scheduling simulation. The newly proposed deadlock avoidance algorithm is detailed, with the proof of its correctness and complexity, followed by the description of an efficient scheduling algorithm, SIMTRAS, which is deduced from the DEADAALG algorithm and has an $O(|S|\cdot|T|^2 \log |T|)$ time complexity. The objective of SIMTRAS, a modified version of DEADAALG, is to minimize average travel times of trains, in addition to detecting and avoiding deadlocks. 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.

LITERATURE REVIEW

Although many authors have discussed deadlock prevention, detection, and avoidance for computer systems, in which preemption is usually an option in order to break a deadlock, this coverage 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 (*3*), as well as its modifications (*4, 5*), is not well adapted to train scheduling, since the algorithms do not guarantee a strategy for deadlock avoidance with an efficient resource reservation mechanism; multiple resource allocation searches are required without any guarantee of the ability to generate train scheduling with deadlock avoidance when such train scheduling exists.

Department of Computer Science and Software Engineering, Concordia University, 1455 De Maisonneuve Boulevard West, Montreal, Quebec H3G 1M8, Canada. Current address for B. Simon: 21 Rue de la Porte de Fer, 45380 La Chapelle Saint Mesmin, France. Corresponding author: B. Jaumard, bjaumard@cse.concordia.ca.

Transportation Research Record: Journal of the Transportation Research Board, No. 2448, Transportation Research Board of the National Academies, Washington, D.C., 2014, pp. 45–52.
DOI: 10.3141/2448-06

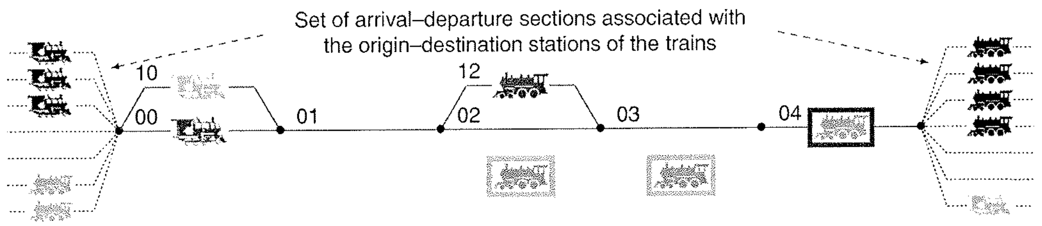

Set of arrival–departure sections associated with the origin–destination stations of the trains
10
00
01
12
02
03
04

FIGURE 1 Drawback of Pachl's method (7).

More recently, Pachl proposed a new deadlock avoidance method, called dynamic route reservation (DRR) (6). The key idea is to consider that when a train requests to leave the section it is traveling on and to move on to the next one, there must be a fast enough check that this movement is safe before it is allowed and a decision on 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 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 modified the DRR reservation process so that if the initial state is solvable, the reservation succeeds (7). However, the new reservation process has a critical drawback: no deadlock detection is included in the algorithm, and consequently when a deadlock occurs, the behavior of the algorithm is not defined. Furthermore, a reservation can be confirmed and later may lead to a deadlock that could have been avoided, and the deadlock issues with arbitrary choices at some iterations remain. An illustration is given in Figure 1, where the green train, initially on Section 04, should not be allowed to travel on Section 02 (see Rule 6, discussed later in the proposed reservation process); however, rules in the Pachl algorithm authorize it.

The algorithm of Pachl is revisited and the reservation mechanism is completely redesigned with four reservation states in order to obtain an accurate deadlock avoidance algorithm (7). In addition, several of the rules of Pachl's algorithm are modified to make use of the four reservation states in order to avoid deadlocks whenever possible and to make sure that the resulting algorithm encounters a deadlock only if there is no alternate way to avoid it (unavoidable deadlock). In summary, the contribution of this study 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, to deduce a train scheduling that is deadlock free in polynomial time whenever one such schedule exists.

DEADLOCK AVOIDANCE AND DETECTION AND TRAIN-SCHEDULING SIMULATORS

A rail system is considered 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. Here it is assumed that sidings are not overlapping. The set of sections is defined as the set of segment sections and siding tracks, including the departure and arrival sections as shown in Figure 1. It is denoted S and indexed by s.

The set of trains is described by a set of eastbound and westbound trains, denoted T and indexed by t, with 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 safe 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 and departure times of all trains at each siding or station.

Train-scheduling simulators are divided into two categories: event driven (8) and time driven (9). Event-driven simulators work 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 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. A new deadlock avoidance and detection algorithm is proposed that can be easily adapted to both event- and time-driven train-scheduling simulators.

NEW DEADLOCK AVOIDANCE AND DETECTION ALGORITHM

An original $O(|S|\cdot|T|)$ deadlock avoidance and detection algorithm is proposed called the DEADAALG algorithm, which dynamically makes section reservations for the trains. If the DEADAALG algorithm successfully completes a sequence of successful reservations for all the trains until their final destinations, a train scheduling without deadlock can be deduced (see the section on the SIMTRAS algorithm); otherwise, an unavoidable deadlock has been identified. The DEADAALG algorithm is an exact algorithm, which is free of the arbitrary selections that lead the algorithm of Pachl to sometimes reach wrong conclusions (7).

Reservation Process

In each section s, an ordered reservation LIST_RESERV(s) is defined in which a new reservation can be inserted in any position but the reservation in the first position must always be released first. At the outset, on section s, 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, which may have four different states:

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

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 the requested state on the sections on which the trains are waiting to move forward (Figure 1) or on the sections on which the trains are running. Consequently, at each iteration, a train is selected (selection rules are discussed later), 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 reservations are 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 change to the confirmed state except the last one, which changes to the requested state. This process means that 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. At any time, there is at most one requested or initiating reservation per section. In the LIST_RESERV(s) reservation files associated with sections, the confirmed reservations are always placed at the beginning and the pending ones at the end.

Train t is occupying siding s—that is, t = occupying (s)—if t has an initiating or a requested reservation for s. If no train has such a reservation, then occupying (s) returns to zero.

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 the booking ahead is successful; meanwhile, the reservation is in a pending state.

Rule 2. If a reservation is requested on section s, which does not contain any pending reservation, a reservation is added in the 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; that is, the latest reservation needs to be confirmed before the previous pending ones. Then the associated train must reserve one section ahead, since there are reservations behind it. This process can only occur on segment sections and is then enforced by Rule 1.

Rule 4. If a reservation is placed behind an initiating reservation, the reservation fails; then there is a circular wait, that is, 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 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 t: the two sections are equivalent; choose, for example, 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 t: the two sections are equivalent; reserve, for example, s_a.

Rule 6d. The siding is occupied by a train running opposite 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 running in the direction opposite t, on s_b: reservation is made on the free siding section, s_a.

Rule 6f. The siding is occupied by one train running 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, for example, s_a.

DEADAALG Algorithm

Next the DEADAALG algorithm is described in detail; thanks to the rules described in the previous section, the algorithm determines whether there exists a feasible schedule without 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, LIST_RESERV(s), and LIST_LOCAL(s), which is local to RESERVATION(t). It calls two functions, RESERVATION(t) and RESERVATION_SEC(s, t), which take care of the reservation of sections for t until DEADAALG either ends successfully or 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 deadlock.

LIST_RESERV($s$) ← $\varnothing$ for all $s \in S$
STATE($s$, $t$) ← $\varnothing$ for all $(s, t) \in S \times T$
$s$ ← section on which $t$ is
STATE($s$, $t$) ← requested for all $t \in T$
▶ ▶ ▶ There cannot be two trains with a requested state on the same section; that is, 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 not, it 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 dow▶ ▶ ▶ Application of Rule 1
  RESERVATION_SEC$(s, t)$
  LIST_LOCAL$(t) \leftarrow$ LIST_LOCAL$(t) \cup \{s\}$
  $s \leftarrow$ next_sec$(s)$

if the end point 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 $s_1$ 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 $s^{\text{selected}}$ be $s_a$
Case 1. Apply Rule 6b with $s^{\text{selected}} \hookrightarrow$ RESERVATION$(t_b)$;
RESERVATION_SEC$(s^{\text{selected}}, t)$

Case 2. Apply any of the other rules with $s^{\text{selected}} \hookrightarrow$ RESERVATION_
SEC$(s^{\text{selected}}, t)$

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

The DEADAALG algorithm is shown in Figure 2, where the westbound black, purple, green, and blue trains are occupying Sections 16, 06, 04, and 14, respectively, and 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 the right depending on whether the train travels westbound or eastbound. Reservation is first sought for the purple train, and it is assumed without loss of generality that it chooses Section 04. It is assumed that contrary to the rules of the DEADAALG algorithm, the blue train does not immediately trigger a reservation process as in Pachl's algorithm. Then the green train has to launch one; this step leads to the situation described in Figure 2*a*. The green train successfully completes its reservation, but then a deadlock is reached, since 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. The DEADAALG algorithm is illustrated in Figure 2*b* and avoids deadlocks since it immediately triggers a reservation process for the blue 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 not, an unavoidable deadlock has been identified. If yes, place the reservation and trigger 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 $t' \neq \varnothing$ 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 $t' \neq \varnothing$ then RESERVATION$(t')$                       ▶ Rule 5
  else Insert $t$ in position $i$ of LIST_RESERV$(s)$                    ▶ Rule 3
```

The failure of the 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. 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.

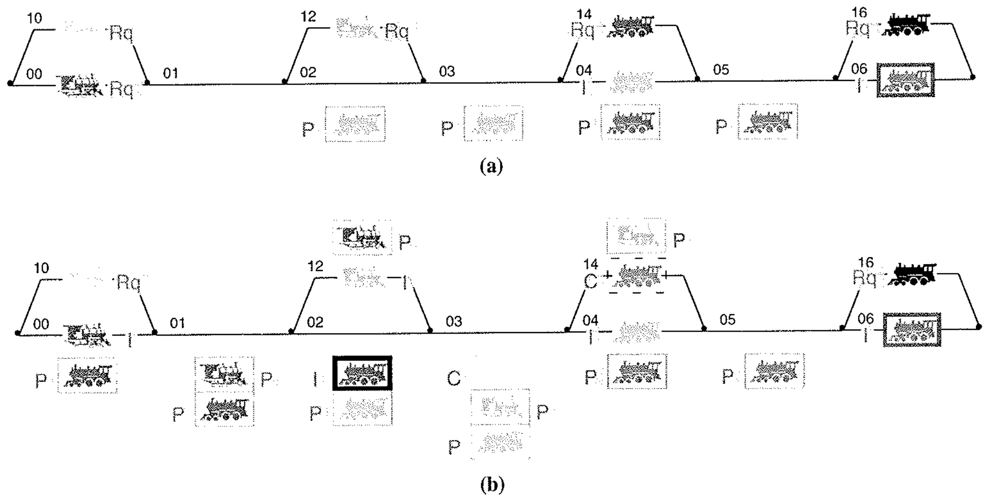


FIGURE 2 DEADAALG algorithm (*Rq* = requested, *I* = initiating, *P* = pending, and *C* = confirmed).

At this point, the red train reserves on Section 01 and its reservation is then placed before the pending one on the blue train. This step causes the yellow train reservation and is located between the blue confirmed and the green pending reservations on Section 03; it concludes on Section 14, which is not occupied (second scheme). Finally, all reservations succeed.

It is easy to derive a train scheduling, called the SIMTRAS algorithm, from the DEADAALG algorithm in the context of an event-driven simulation tool by 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 is in the first position) is released and suppressed. If the end point of the new section is the final destination of t, t has successfully reached it.

The detailed description of the SIMTRAS algorithm is provided next.

SIMTRAS Scheduling Algorithm

```
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 traveling time
Add multiple parallel sections at the origin/destination stations of the trains
$L_s \leftarrow 0$ for $s \in S$; STATE $(s, t) \leftarrow 0$; DEPART$(s, t) \leftarrow 0$ for $(s, t) \in S \times T$
for $t \in T$ do
    $s \leftarrow$ departure section of $t$ (one extremity)
    STATE$(s, t) \leftarrow$ requested; 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: for all $t'$, DEPART$(s(t), t) \leq$ DEPART$(s(t'), t')$, where $t$ = occupying $(s(t))$
            ▶ If $t$ is on a siding track, and a reservation of an opposite train prevents it from leaving at DEPART$(s(t), t)$,
              we associate the minimal time at which it can leave instead of DEPART$(s(t), t)$.
            ▶ Ties can be broken (e.g., by selecting the easternmost siding $s$) and then for remaining ties, with the selection of eastbound train.
        RESERVATION $(t)$
    ▶ The reservation is repeated 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 DEPART $(s, t)$
        ▶To be added at the end of RESERVATION
    DEPART$t(s^{init}, t) \leftarrow \min \{\tau \geq$ DEPART$(s^{init}, t)$: $t$ can leave $s^{init}$ at $\tau$ and reach $s$ selected with no stop$\}$ for $s \in L^{process} \cup \{s^{selected}\}$ do
        DEPART$(s, t) \leftarrow$ DEPART$(s^{init}, t)$ + travel time from the end of $s^{init}$ to the end of $s$
        Change the position of $t$ in $L_s$ so that the list remains ordered by increasing DEPART$(s, t)$.
```

DEADAALG ALGORITHM PERFORMANCE AND COMPLEXITY

Because of the lack of space, an outline of the correctness and complexity of the DEADAALG algorithm is provided in the case of sidings with two alternate tracks; the detailed proof can be found elsewhere (*10*).

Theorem 1 The DEADAALG algorithm is finite. With an $O(|S|\cdot|T|)$ 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—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 end points of the line network (Figure 1). A train departure plan is solvable if all the trains can reach their destination without encountering a deadlock. Within the DEADAALG algorithm, a train configuration can be reached in which all the reservations are in the confirmed state except on one track section of its destination. For the RESERVATION function, at each iteration, the train configuration output by the RESERVATION function is such that each train has a unique requested reservation state and the last reservation is for the section on which it is located. The first step of the proof consists in demonstrating that if RESERVATION(t) is launched on a solvable train configuration and concludes, it is possible to reach a train configuration that is output by RESERVATION(t) throughout iterative applications of Scheduling Rule 1 (see end of the previous section), 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, 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 by using Scheduling Rule 1.

Step 3. If a RESERVATION succeeds, 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.

Once those three demonstration steps are completed, it can be concluded, on the basis of Step 3, that the output train configuration is solvable. In this way, the focus is not on the trains involved in the RESERVATION function call, but only on the set of spanned track sections, and it can be proved to be in a safe situation; that is, 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 two steps. From now on, the function MODIF is considered, which neglects 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 other functions successfully. 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, it can be concluded that if RESERVATION fails, all the MODIF function calls fail, and the case of Step 5 is reached (by using the previous result) to show that all choices lead to a deadlock, so the input train configuration was not solvable. In this way, it is shown that the choices made in RESERVATION are relevant: if there is a solution, one is found. The DEADAALG algorithm works as follows. While there is a train in the system, one is chosen (the order will only modify the associated scheduling; see the next section), and RESERVATION is called on it. If it succeeds, the new train configuration is continued. Otherwise, the train departure plan was a deadlock.

Proof of Complexity

RESERVATION_SEC(s, t) cannot be called twice on the same pair (s, t), so it cannot be called more than $|S|\cdot|T|$ times. If LIST_RESERV(s) is implemented with double linked lists and the pointers to the last train are kept in the confirmed state in each list, a complexity of $O(|S|\cdot|T|)$ is obtained.

Proof of Correction

If the train departure plan is solvable, the first call to RESERVATION succeeds and leads to a solvable train configuration. Then, by using induction, all the RESERVATION calls succeed and lead to solvable train configurations. Using Scheduling Rule 1, a scheduling free of deadlock can be shown. 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 AND DETECTION

It was explained earlier how to derive a first train scheduling from the output of the DEADAALG algorithm; Theorem 1 has proved this to be correct. Its improvement with respect to the minimization of the average travel times of the trains is now discussed.

While DEADAALG is applied, the departure time of train t on section s, denoted by DEPART(s, t), can be computed in polynomial time by using the section average travel times (with the data provided by Canadian Pacific Railway). 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 (which belongs either to a siding or to a segment).

Assuming that the objective is to minimize the average travel times of the trains, at each step train t is selected with the smallest DEPART(s, t) on the s section it is requesting. In this way, the distribution of the requested sections at each step is the closest to a snapshot of the future scheduling: it behaves as a greedy algorithm: at each 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), the minimal tardiness is used. In 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 procedure adds an $O(\log|T|)$ factor to the complexity.

In addition, to avoid the formation of queues from congestion, the following feature is added. 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 a queue is encountered, 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 the case of one train in each direction that monopolizes the bottleneck alternatively is avoided.

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, a check is made whether the reservations can be placed sooner in the reservation lists. The drawback of this modification is an increased time complexity of the algorithm, because a nonconstant part of the LIST_RESERV has to be scanned to find the minimum possible time. A basic implementation leads to an $O(|S|\cdot|T|^2 \log|T|)$ complexity.

In order to process segment data with average travel times that vary and depend on the direction (east versus west, going up versus going down, or empty cars versus loaded cars), each segment is modeled by a section. Two trains going in the same direction are allowed to be initially on the same section if the differences in 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

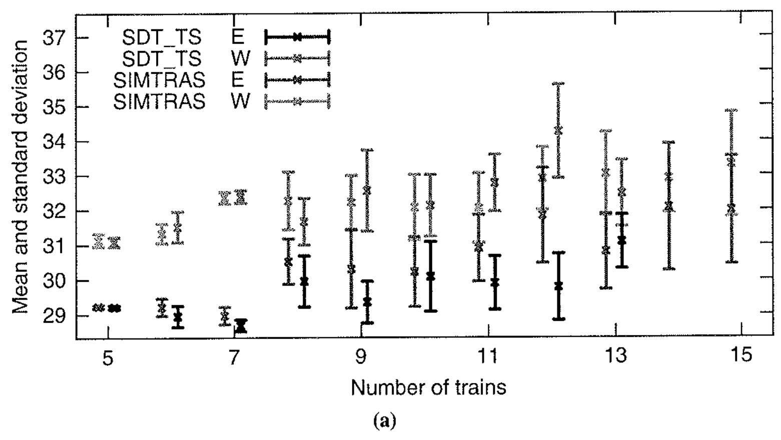
All the algorithms and functions described in the previous section were implemented in C++.

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) was evaluated. It is a single-track railway system, with 77 sidings or stations. In terms of capacity (number of alternate tracks), one alternate track is assumed at every station or siding.

A set of 8 to 30 trains is used, with the same number of trains from Vancouver toward Calgary as from Calgary toward Vancouver. Departure times are uniformly distributed over a time period of 24 h when a 24-h planning period is considered, and over a time period of $24(1-1/|T|)$ h for planning periods spanning several days (60 in these 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 s for 1,000 trains and about 40 s for 2,000 trains.

In Figures 3 and 4, the average travel times and their standard deviations of the overall daily fleet of trains obtained with the SIMTRAS algorithm are plotted for different numbers of daily trains (from 5 to 15 over a 1-day time period in Figure 3 and from 11 to 15 over a 60-day time period in Figure 4). The eastbound trains are distinguished from the westbound trains.



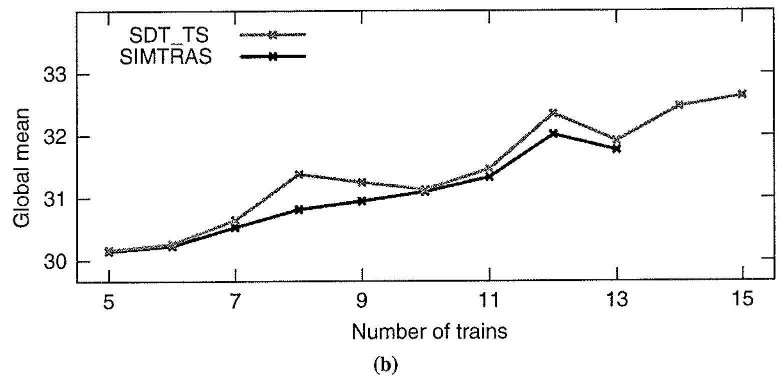
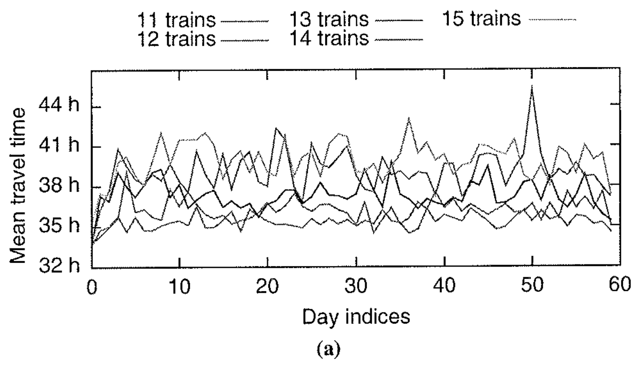


FIGURE 3 Average travel times for train scheduling output by SDT_TS and SIMTRAS for algorithm 1-day horizon.



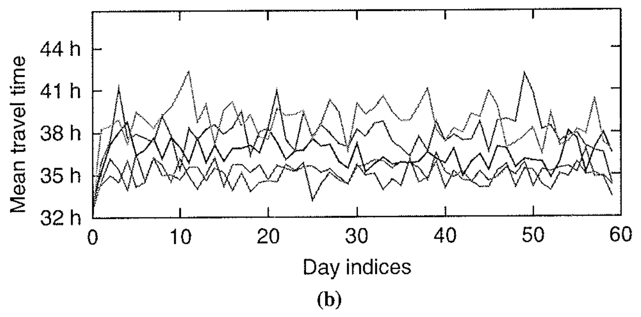


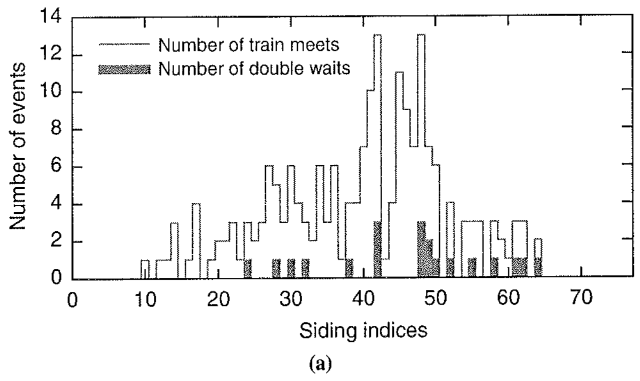
FIGURE 4 Average travel times for train scheduling output by SDT_TS and SIMTRAS algorithm for 60-day horizon.

In addition, in order to evaluate the quality of the train timetables output by the SIMTRAS algorithm, the results obtained by the ε-optimal SDT_TS algorithm of Jaumard et al. (*11*) were added. Light blue is for eastbound trains and dark blue for westbound trains.

Taking into account the standard deviations measured over a 1-day time period, it is observed that the average travel times are fairly stable over a 60-day time period; that is, there is no deterioration of the average travel times. Longer travel times for the westbound trains are 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,* the standard deviations of the SIMTRAS and SDT_TS algorithms are fairly similar, and 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*). Therein the 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 it is believed that this is one of the first times, or the first time, that such differences have been measured. 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 5 the siding use is investigated first over a 1-day period (Figure 5*a*) and then over a 60-day period (Figure 5*b*). The red curve indicates the number of train meets. While over a 1-day period sidings close to the origin or the destination are not used, the use



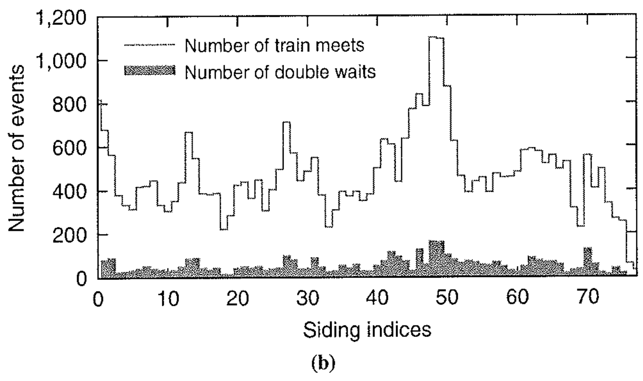


FIGURE 5 Siding use: (*a*) 14 trains, 1 day, and (*b*) 14 trains, 60 days.

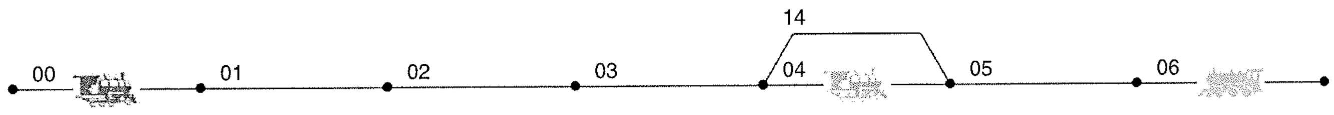


FIGURE 6 Double wait that leads to deadlock avoidance.

of the sidings is much more uniform over a 60-day time period, although there are differences in their use. The green curve represents the number of train meets associated with double waits; this curve indicates that some trains need to wait on any of the tracks of the sidings (Figure 6). In Figure 6, the blue train waits on the south Track 04 of the siding, and later the green train will need to wait on the north Track 14 of the siding in order to avoid a deadlock.

CONCLUSION

A first exact ($|S|\cdot|T|^2\log|T|$) and highly scalable deadlock detection and avoidance DEADAALG algorithm is proposed for train scheduling. In addition, the DEADAALG algorithm favorably competes with the SDT_TS exact algorithm of Pachl (*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 two alternate tracks.

REFERENCES

1. Toueg, S., and K. Steiglitz. Some Complexity Results in the Design of Deadlock-free Packet Switching Networks. *SIAM Journal on Computing,* Vol. 10, No. 4, 1981, pp. 702–712.
2. Sahin, I. Railway Traffic Control and Train Scheduling Based on Inter-train Conflict Management. *Transportation Research Part B: Methodological,* Vol. 33, 1999, pp. 511–534.
3. Dijkstra, E.W. *Cooperating Sequential Processes.* Technical Report EWD-113. Technical University, Eindhoven, Netherlands, 1965.
4. Belik, F. Deadlock Avoidance with a Modified Banker's Algorithm. *BIT,* Vol. 27, 1987, pp. 290–305.
5. Belik, F. An Efficient Deadlock Avoidance Technique. *IEEE Transactions on Computers,* Vol. 39, 1990, pp. 882–888.
6. Pachl, J. Avoiding Deadlocks in Synchronous Railway Simulations. Presented at 2nd International Seminar on Railway Operations Modelling and Analysis, Hannover, Germany, 2007.
7. Pachl, J. Deadlock Avoidance in Railroad Operations Simulations. Presented at 90th Annual Meeting of the Transportation Research Board, Washington, D.C., 2011.
8. Grube, P., F. Núnez, 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,* Vol. 19, 2011, pp. 393–405.
9. Li, F., Z. Gao, K. Li, and L. Yang. Efficient Scheduling of Railway Traffic Based on Global Information of Train. *Transportation Research Part B: Methodological,* Vol. 42, 2008, pp. 1008–1030.
10. Simon, B., B. Jaumard, and T. H. Le. *Deadlock Avoidance and Detection in Railway Simulation Systems.* Technical Report G-2013-43. Group for Research in Decision Analysis, Montreal, Quebec, Canada, 2013. http://www.gerad.ca/en/publications/cahiers_rech.php.
11. Jaumard, B., T. H. Le, H. Tian, A. Akgunduz, and P. Finnie. An Enhanced Optimization Model for Scheduling Freight Trains. *Proc., Joint Rail Conference,* Knoxville, Tenn., American Society of Mechanical Engineers, 2013, pp. 1–10.

The Railroad Operating Technologies Committee peer-reviewed this paper.