Modeling Train Movements Through Complex Rail Networks

QUAN LU and MAGED DESSOUKY
University of Southern California
and
ROBERT C. LEACHMAN
University of California, Berkeley

Trains operating in densely populated metropolitan areas typically encounter complex trackage configurations. To make optimal use of the available rail capacity, some portions of the rail network may consist of single-track lines while other locations may consist of double- or triple-track lines. Because of varying local conditions, different points in the rail network may have different speed limits. We formulate a graphical technique for modeling such complex rail networks; and we use this technique to develop a deadlock-free algorithm for dispatching each train to its destination with nearly minimal travel time while (a) abiding by the speed limits at each point on each train's route, and (b) maintaining adequate headways between trains. We implemented this train-dispatching algorithm in a simulation model of the movements of passenger and freight trains in Los Angeles County, and we validated the simulation as yielding an adequate approximation to the current system performance.

Categories and Subject Descriptors: I.6.3 [**Simulation and Modeling**]: Applications; I.6.5 [**Simulation and Modeling**]: Modeling Development—*Modeling methodologies*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Trains, modeling, dispatching, deadlock

1. INTRODUCTION

Worldwide container trade is growing at a 9.5% annual rate, and the U.S. rate is around 6% (Vickerman [1998]). For example, the Ports of Los Angeles and Long Beach (San Pedro Bay Ports) are anticipated to double and possibly triple their cargo by 2020. The growth in the number of containers has already introduced congestion and threatened the accessibility and capacity of the rail network system in the Los Angeles area.

The research reported in this paper was partially supported by the National Science Foundation under grant DMI-0307500.
Authors' addresses: Q. Lu and M. Dessouky, Department of Industrial and Systems Engineering, University of Southern California, Los Angeles, CA 90089-0193; email: qlu@scf.usc.edu, maged@usc.edu; R. C. Leachman, Department of Industrial and Operations Research, University of California, Berkeley, Berkeley, CA 94720; email: leachman@ieor.berkeley.edu.

To partially address this rail traffic growth, the Alameda Corridor is being developed in Los Angeles County. The Alameda Corridor is a high-speed double-track line from the Ports of Long Beach and Los Angeles to Downtown Los Angeles (Leachman [1991]). However, the Alameda Corridor does not address the transcontinental rail traffic from Downtown Los Angeles to the Eastern Inland area.

The rail network in this area is rather complex. Some high-traffic portions of the rail network in this area consist of double- or triple-track lines. The control logic of rail networks that consist of multiple trackage can be very complicated in order to avoid potential train deadlock movements. For example, a rail network consisting strictly of double-track lines is simple to control since it is similar to vehicle traffic movement. That is, a control logic of keeping the traffic moving only on the right-hand side ensures there will be no deadlocks. No similar simple rules exist when there is a mix of trackage configurations in the rail network. In fact, for a general trackage network, to determine the optimal dispatch times that minimize total train delays is an NP-hard problem, because it can be reduced to the deadlock avoidance problem for sequential resource allocation, which has been proven to be NP-hard by Lawley and Reveliotis [2001].

Another complicating factor of rail networks in metropolitan areas is the existence of multiple speed limits at different points in the network because of physical contours, crossovers, or other safety considerations. In this case, the issue is to determine the fastest speed the train can travel at each instant of time without violating any speed limit considering the train's acceleration and deceleration rates. If the acceleration and deceleration rates are assumed to be infinite, the speed of the train at each instant of time is simply set to the constraining speed limit. When they are finite, we show that the fastest speed the train can travel at each instant of time is a complex combination of the acceleration and deceleration rates and the uniform motion of the train.

In order to analyze the capacity of the rail network in the area from Downtown Los Angeles to the Eastern Inland, we developed a deadlock-free simulation methodology to model rail networks that consist of a multiple trackage configurations and speed limits. The purpose of this paper is to present this methodology. Our primary contributions are the development of a network framework that describes the trains and their travel paths, the development of an algorithm that determines the fastest a train can travel while observing multiple speed limits, and the development of a deadlock-free dispatching heuristic.

There has been some prior work in simulation modeling of rail networks. Dessouky and Leachman [1995] developed a simulation modeling methodology for either strictly single-track or double-track rail networks consisting of a single speed limit without considering deceleration rates. This prior model was suitable for simulating the Alameda Corridor, which is a high-speed double-track network. Since there was a single speed limit and a small number of locations for potential train conflicts (e.g., crossover junctions), there was no need to take into account the train deceleration process. Petersen and Taylor [1982] presented a structured model for rail line simulation. They divided the line into track segments representing the stretches of track between adjacent

switches and developed algebraic relationships to represent the model logic. Higgins and Kozan [1998] proposed an analytical model to quantify the positive delay for individual passenger trains on the track links in an urban rail network. They also developed a schedule time-driven simulation model with fixed routing to validate their analytical model. Cheng [1998] applied a hybrid approach consisting of a network-based simulation and an event-driven simulation model of rail networks with tracks dedicated to the trains running in the same direction.

Carey [1994] developed a mathematical model for routing trains under different speed limits. He assumed that the departure schedule of the trains from the initial station is deterministic. Komaya [1991] used a knowledge-based model to simulate train movements for large and complicated rail systems. Lewellen and Tumay [1998] described a model used to simulate train movements for Union Pacific. However, the authors did not present any details of the model logic.

The modeling methodology presented in this paper differs from the previous work by considering a multiple trackage configurations in the same rail network with multiple speed limits while taking into account the trains' acceleration and deceleration rates. Furthermore, we do not assume that the initial departure schedule of the trains is known. We model it as a stochastic process for freight trains and as a fixed schedule for passenger trains.

Imbedded in our modeling methodology is a central dispatching algorithm that decides the movement of each train in the network considering whether to continue moving at the same speed, to accelerate or decelerate, or to stop. The algorithm also selects the next track to seize among multiple alternative tracks. Our central dispatching heuristic guarantees that no deadlock occurs while attempting to keep the train delays to a minimum. Even though the proposed simulation modeling methodology was applied to the rail network from Downtown Los Angeles to the Eastern Inland area, it can be applied to various situations to simulate rail networks with any kinds of topology, crossovers, and speed limits.

The rest of the paper is organized as follows. Section 2 gives the details of the system model and the train movement control logic. Section 3 formulates and solves the problem of determining the fastest speed a train can travel under multiple speed limits while considering the train's acceleration and deceleration rates. Section 4 describes the issues of deadlock avoidance and efficient routing. Finally, in Section 5 we present the overall integrated model and show its effectiveness in modeling train movement in Los Angeles County.

2. DESCRIPTION OF SYSTEM MODEL

This section presents a simulation modeling methodology used to analyze generic complex rail networks. The modeling methodology does not depend on the size of the rail network and is insensitive to the trackage configuration. That is, it can be used to model rail networks consisting of single-track lines, double-track lines, or any number of tracks. Note that trains operate quite differently depending on the number of tracks. For example, in a single-track

system, trains moving in opposite directions compete for the same track and there must exist sufficient buffer trackage for trains to wait for opposing direction trains to move in order to prevent potential deadlocks. In a double-track system, the common routing logic is to dedicate each track for trains moving in only one direction. A triple-track system exhibits features similar to both single-track and double-track systems.

We will first discuss how to construct the simulation network from the actual physical system. Then we will discuss the train movement process in our simulation network. Although the train movement process is a continuous process, our modeling approach is a discrete-event methodology. We approximate the continuous motion of train movement by dividing the movement into small discrete steps. A continuous simulation modeling approach would require the development of motion equations that represent train movement, which are difficult to represent mathematically in the context of train conflicts and deadlock-free movement.

2.1 Simulation Network Construction

The physical resources that we model are: rail junctions and track segments. A rail junction is typically used for train crossover movement in a rail network. One idea behind the modeling approach is to divide the physical track into segments, as in Dessouky and Leachman [1995]. A segment is the minimum unit in the simulation model and each segment is represented as a unique resource with capacity *one*. Junctions are also represented as a resource with capacity one in our simulation model. A track segment has the following characteristics:

(1) Travel in each segment is restricted by one speed limit.
(2) No junction exists within the segment. Junctions can only be located at the beginning or end of a segment.
(3) The length of the segment is no longer than the maximum train length.

The first two characteristics are not restrictive since there is no limit on the minimum length of the segment. Hence, the definition of the segment is sufficiently generic to model any physical trackage configuration. However, having many small track segments will increase the number of resources in the simulation model and the computational run time of the model. On the other hand, since we restrict the capacity of each segment to be one, too large of a segment definition will increase the headway between trains, needlessly decreasing the capacity of the network. Thus, the third characteristic restricts the maximum size of the segment to be the maximum train length. The above three characteristics can also be considered as a rule to divide the segments. We refer it as the *segment divide rule*.

As an illustration of the segment definition, Figure 1 diagrams a small portion of a rail network near Downtown Los Angeles. There are six junctions in the network at points B, F, H, I, K, and L. Note that junctions F and K consist of a double-direction crossover. In Figure 1, we represent this type of crossover with two cross connections. Junctions K and L consist of a single-direction crossover. In Figure 1, we represent this type of crossover with a one cross connection.

Junction Speed Limit

Name	Speed limit
B	20 miles/hour
F	20 miles/hour
H	20 miles/hour
I	20 miles/hour
K	20 miles/hour
L	20 miles/hour

Segment Speed Limit

Name	Length	Speed limit
A-B	1.4 miles	25 miles/hour
B-C	1.1 miles	50 miles/hour
B-I	0.3 miles	20 miles/hour
D-E	1 mile	50 miles/hour
E-F	0.5 miles	20 miles/hour
F-G	1.5 miles	40 miles/hour
H-I	1.5 miles	40 miles/hour
H-J	0.7 miles	20 miles/hour
I-K	0.5 miles	30 miles/hour
J-L	1.5 miles	20 miles/hour
K-M	1.5 miles	40 miles/hour

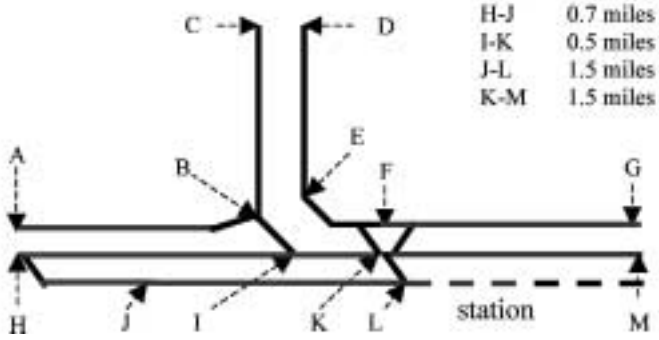

Fig. 1. A sample rail network.

The speed limit at all these junctions is assumed to be 20 miles/hour. Assuming the maximum train length is 1.5 miles, we define eleven track segments listed in Figure 1. Each segment has a single speed limit and is a resource. Only one train can be in a track segment at each instant of time. Furthermore, a train at segment E-F crossing over the track to move to segment K-M will seize junction resources F and K, thus blocking all the traffic that needs junction resources F and K (e.g., traffic from segment F-G to segment I-K).

We next translate the segment definition of the physical system into a network architecture. Each node in our network defines a combination of one or more contiguous segments. Each node has two ports: port 0 and port 1. Port 0 indicates the starting point of travel for a train moving in the node from one direction. Port 1 indicates the starting point of travel in the opposite direction of port 0. Two distances locate each segment in a node:

(1) the distance from the end of the segment to port 1 of this node, and
(2) the length of the segment itself.

Note that the length of the node equals the sum of all the segments' lengths within this node.

The nodes are connected by arcs, which represent movement from one node to another. Arcs may include junctions or not. All the arcs in our network are undirected and have zero length. Therefore, the total travel distance of a train in the network equals the sum of the lengths of the nodes it visits. As we will describe in detail in Section 2.2, in order to move to a successor node, all the resources associated with the track segments of the successor node, as well as any connecting junction resources, must be available. Since the capacities of the track segment and junction resources are unity, only one train can be at

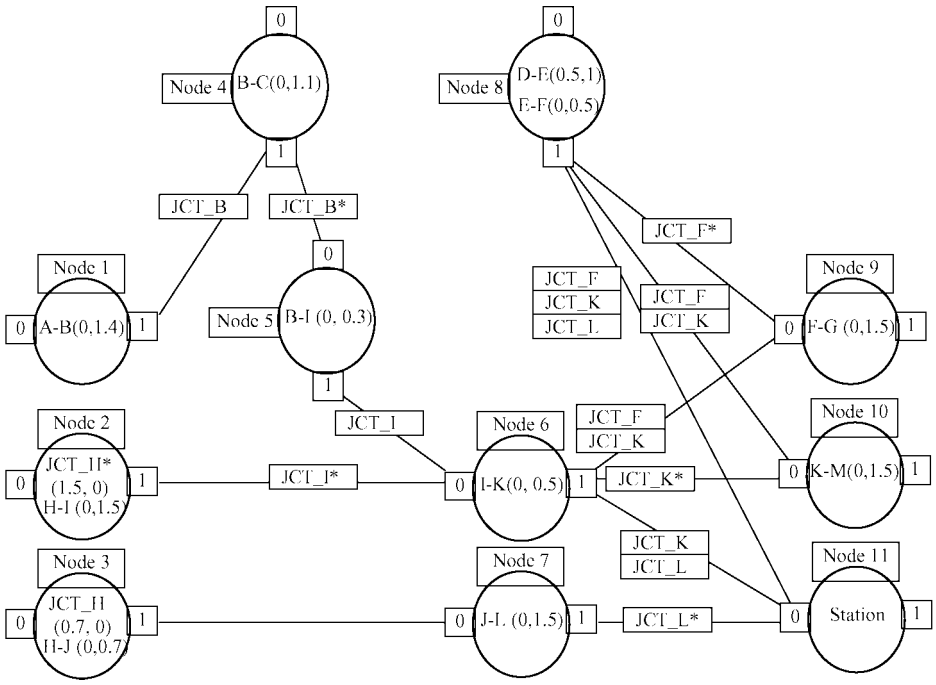

Fig. 2. Network architecture for the sample rail network.

a node at any instant of time. Figure 2 illustrates the network concept for the example given in Figure 1.

In each node, the two numbers within the parentheses next to the segment's name denote the distance from the end of this segment to port 1 of this node, and the segment's length, respectively. For instance, node 8 contains two track segments, D-E and E-F, from port 0 to port 1. Segment D-E is 1 mile long. Since segment E-F is between segment D-E and port 1 of node 8, the length from the end of segment D-E to port 1 of node 8 equals the length of segment E-F, 0.5 miles. Similarly, segment E-F is 0.5 miles long and, because there are no other segments between it and port 1, the distance from the end of segment E-F to port 1 of node 8 is 0.

One characteristic of our network definition is the existence of multiple paths for each origin and destination node pair. For instance, a train starting from point H and ending at the station has two alternatives in Figure 1. Alternative 1 consists of moving to node 2, then to node 6, and then to node 11 (i.e., H-I to I-K and crossover at junctions K and L to the station). Alternative 2 consists of moving to node 3, then to node 7, and then to node 11 (i.e., H-J to J-L and to the station). Besides the different speed limits, the total distances of these two paths are also different. Our network structure allows for flexible routing once the train enters the network. That is, trains with the same origin and destination may run through different paths according to the routing logic discussed in Section 4.

Arcs connect nodes. Some arcs include junction resources. These arc-included junctions are necessary since they represent the junction resources the train needs in order to travel from one node to another. For instance, a train that is trying to enter the station from node 8 needs to cross at junctions F, K, and L. In the network, the train moving from port 1 of node 8, crossing the arc connected between node 8 and node 11, and finally entering node 11, represents this motion. The resources that need to be seized in this movement in sequence are junction F, junction K, junction L, and the station. Sometimes, we also separate segments in different nodes and connect them with an arc containing no junction resources. Including both of these two segments in a node may result in an unnecessarily high spacing between trains, since only one train can occupy a node at a time. For example, segment H-J (node 3) is connected to segment J-L (node 7) by an arc including no junction resource.

The speed limits for the segments are applied to all trains traveling across these segments. Unlike the speed limit for the segment, the speed limit for the junction is only used to restrict the trains that *cross over* at the junction. If a train only bypasses the junction, the speed limit for the junction will not be applied. For instance, a train moving from segment B-I to segment I-K needs to reduce its speed to less than 20 miles/hour while passing through junction I. But a train moving from segment H-I to segment I-K does not need to be restricted by junction I's speed limit. To distinguish this difference, in our network we use an "*" following a junction name to denote that this junction is needed for the train passing it, but no junction speed limit needs to be applied.

Figure 2 shows the advantages of defining the ports for a node. First, they indicate the direction of train movement when it enters a node. Second, in reality, a train in segment H-I cannot turn around and move to segment B-I at junction I. But from the network point of view, a train could move from node 2 to node 5 through node 6, since node 2, node 5, and node 6 are all connected. The definition of ports solves this conflict. We add a restriction that if a train enters a node from port 0 (respectively, port 1), then the train must leave from port 1 (respectively, port 0).

2.2 Train Movement Process

The entities in the simulation model are the trains. To support the ability of flexibly routing train entities, we define a routing table at each node that stores a sorted node list for each route, which lists all the successor nodes that have at least one path to the train's destination node. The successor nodes in a list are sorted in ascending order of the minimum travel time from the current node to the destination node. The minimum travel time is computed assuming there is no downstream conflicting traffic ahead of the current train. Note that the shortest path in terms of travel distance may not be the fastest path in terms of travel time, because of the possible existence of different speed limits for the different paths. The routing tables at nodes are set automatically before the simulation starts and remain unchanged through the entire simulation. Note that, in practice, the routing tables have to be updated in real time to

reflect congestion, accidents, and mechanical malfunctions that may render some segments in the network unusable.

A successor node is considered *available* if the following two conditions are met:

(1) All track segments' resources of the successor node must be currently available as well as any possible connecting junction resources.
(2) The movement of a train to this successor node must not create a deadlock.

When a train enters a node, a routing algorithm is applied to check the availability of each successor node and locate the list of the *available* successor nodes. Then, one of the available successor nodes will be selected for the train's next move based on some heuristic criteria. The complete details of the routing algorithm and successor node selection criteria are presented in Section 4. If no successor node is currently available, the train will have to stop to wait for an available node.

In this paper, we assume that each train has a constant acceleration rate a_1 and deceleration rate a_2. Note that different trains may have different a_1 and a_2.

We cannot wait until the head of the train reaches the end of the node to make a decision on whether a train should stop or not since we need to account for decelerating time, and some distance is necessary for a running train to fully stop. A routing algorithm (see the details in Section 4) is applied before the head of the train reaches the end of each node it passes. The point of applying the routing algorithm to decide whether a train should stop or not is referred to as the *stop checking point* (SCP) of the node. Let S_i be the distance from the SCP of node i to the end of node i and V_i be the train's velocity at the SCP of node i. The relationship between S_i and V_i is (Bueche et. al. [1997])

$$V_i = \sqrt{2a_2 S_i} \tag{2.1}$$

Figure 3 illustrates the concept of a stop checking point. Suppose a train is moving from node 8 to node 10 with the head of the train currently at position E in node 8. The velocity of the train at position E is 20 miles/hour. The SCP at node 10 is set to be the intersection of the two dotted lines. Each point on the thick dotted line represents the fastest speed the train can travel at that location, while the train is moving from node 8 to node 10 without stopping. Each point on the thin dotted line represents the fastest speed the train can travel if the train needs to fully stop at the end of node 10 (point M). Note that the train has to adhere to a speed limit of 20 miles/hour even after its head has passed point K and is in the zone with a 40 miles/hour speed limit, because the maximum speed of the train at each instant of time is always restricted by the minimum speed limit of all the tracks it is occupying. In this case, the speed limit at segment E-F still restricts the train before its tail passes point F (its head arrives at point P).

We next define the different types of events used in our simulation system, the logic of the train movement process, and the method for seizing and freeing the resources within the nodes (track segments) and arcs (junctions). Four types

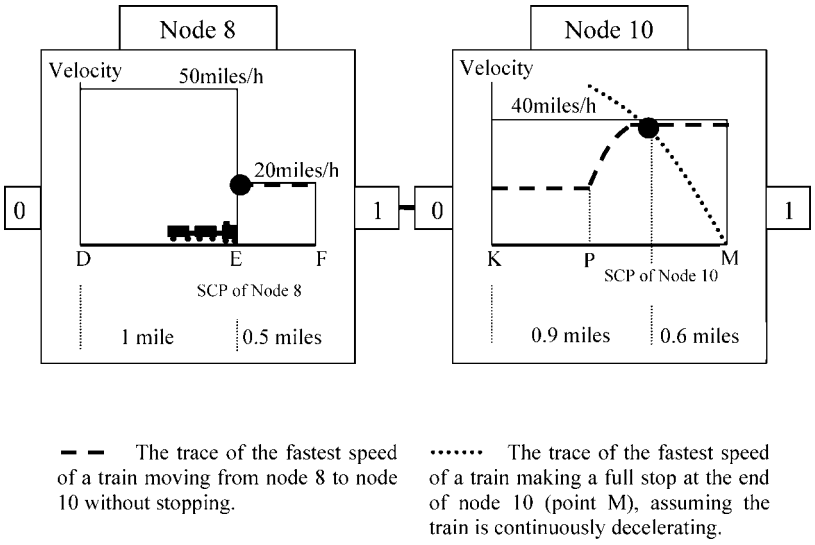

Fig. 3. Stop checking point example.

of events are defined:

(1) arrival of a train to a SCP,
(2) freeing of a resource,
(3) a train coming to a full stop, and
(4) arrival of a train to a terminal station.

If train m's head reaches the SCP of node i, the arrival SCP event is invoked. Then the central dispatching algorithm is called to determine train m's next movement. If the central dispatching algorithm finds that there exists one or more available successor nodes for train m to move forward from the current node i, the algorithm selects one of them, let it be j, and commands train m to move forward to node j. (Details on the selection criteria if more than one successor node is available are provided in Section 4.) Then, train m performs the following operations in sequence:

(1) Let train m seize all the resources belonging to node j as well the arc connecting node i and node j.
(2) Determine S_j and V_j of the SCP of node j.
(3) Calculate the minimum travel time t subject to the speed-limit constraints, while train m's head moves from the SCP of node i to the SCP of node j. This step is based on the Velocity_Augmenting algorithm described in Section 3. Schedule an arrival event of train m at the SCP of node j after t time units from the current time.
(4) Schedule the resource-freeing events to free all the junction and segment resources that train m has currently seized and will pass through during the movement from the SCP of node i to the SCP of node j. The exact times to signal these events are computed based on the algorithm in Section 3.

If the central dispatching algorithm finds that there is no available successor node for train m to move forward, it returns a command to decelerate train m immediately to prepare to stop at the end of node i. However, this does not imply that train m will necessarily stop, since a successor node may become available before train m fully stops at the end of node i. We use two separate queues to distinguish trains in the deceleration process from the trains that have come to a complete stop. Queue 1 stores all the train entities that are in the deceleration process between the SCP and the end of the node. Queue 2 stores all the train entities that have come to a complete stop at the end of the nodes. In order to minimize train stoppage, we give priority to trains in queue 1 over trains in queue 2 when a resource is freed. If train m is notified to prepare to stop at the SCP of node i, the following three operations are performed.

(1) Store train m in Queue 1.
(2) Schedule an event for train m to come to a complete stop at the end of node i after $\sqrt{2S_i/a_2}$ time units from the current time.
(3) Schedule all the resource-freeing events of all the resources that can be freed while train m moves from the SCP of node i to the end of node i.

When a resource-freeing event occurs, a routing algorithm is invoked to find a decelerating train in queue 1, which can stop decelerating and start accelerating after seizing the currently freed resource. If such a train is found, let it be train m. First, the scheduled event of train m coming to a complete stop at the end of node i is removed from the event calendar. Also, the scheduled events of freeing the resources that train m will release before its arrival at the end of node i will be rescheduled, since train m has changed its speed. Finally, the entity of train m is removed from queue 1. Then, train m seizes all the resources it needs and starts moving to the SCP of the new available successor node. If no such train m is identified in queue 1 that can stop decelerating, the routing algorithm checks if a train in queue 2 can begin moving from the stop state. The train entities in queue 2 are sorted by some priority based on the trains' characteristics (e.g., a passenger train has higher priority than a freight train). If a train in queue 2 is identified and can begin moving to its successor node, the train will be removed from queue 2, and the train will seize all the resources it needs and start moving to the SCP of the new available successor node.

If an event associated with a train coming to a full stop occurs, its associated entity in queue 1 is moved to queue 2. When a train arrives at a terminal station, the train entity is terminated and the statistical information regarding trip time and delay is collected.

3. MINIMUM RUN TIMES UNDER MULTIPLE SPEED LIMITS

In rural areas, a single speed limit may apply for long stretches of travel. In contrast, many rail networks in metropolitan areas consist of multiple speed limits because of physical contours, crossovers, or other safety considerations. In this section, we develop an algorithm that determines the minimum run times of trains traveling on the segments and junctions with multiple speed limits considering maximum train speed, and acceleration and deceleration rates. In

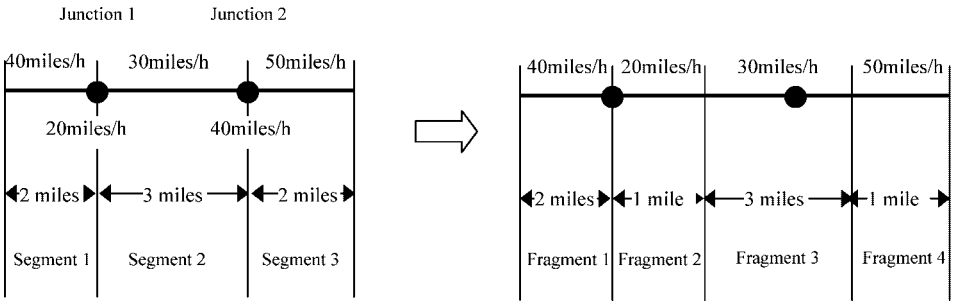

Fig. 4. Fragment generation based on speed limits. h = hour.

our simulation methodology, each track segment and junction has a single speed limit. Note that the effective distance of any junction speed limit on the train equals the length of the train, and the effective distance of any segment speed limit equals the length of the segment plus the length of the train. Therefore, at each point in the rail network, multiple speed limits may constrain the speed the train can travel. In order to identify the constraining speed limit and have a single speed limit apply at each point in the rail network, we introduce the concept of fragments. We define a *fragment* as a portion of the track, which has the same speed limit equal to the minimum junction/segment speed limit applied to it. Any contiguous fragments' speed limits must be different.

The translation of the segments to fragments is performed automatically each time before the minimum run times are computed. Figure 4 gives an example of this translation. In this example, we have three segments with speed limits of 40, 30, and 50 miles/hour and two crossover junctions with speed limits of 20 and 40 miles/hour. Assume that train *m* is 1 mile long and is moving from left to right in the left diagram in Figure 4. The diagram on the right in Figure 4 shows the resulting fragments. The minimal speed limit of the first 2 miles is restricted by the speed limit of segment 1, which equals 40 miles/hour. The minimal speed limit of the next 1 mile is restricted by the speed limit of junction 1, which equals 20 miles/hour. Thus, fragment 1 between the first and second mileposts is created with a single speed limit of 40 miles/hour, and fragment 2 between the second and third mileposts is created with a single speed limit of 20 miles/hour. When train *m*'s head is in the last two miles of segment 2, the effective speed limit equals the speed limit of segment 2, which is 30 miles/hour. When train *m*'s head moves beyond junction 2 and is still within the first mile of segment 3, the following three speed limits are simultaneously applied to train *m*: the speed limit of segment 2 (train *m*'s tail is still in segment 2), the speed limit of junction 2 (train *m* is passing junction 2), and the speed limit of segment 3 (train *m*'s head is within segment 3). The effective speed limit equals the minimal of these three speed limits, which is 30 miles/hour. Thus, we create fragment 3, which has a single speed limit of 30 miles/hour and covers the last 2 miles of segment 2 as well as the first mile of segment 3. When train *m*'s head is traveling in the last mile of segment 3, train *m* is only restricted by the speed limit of segment 3. Fragment 4 is created to represent this single speed limit portion.

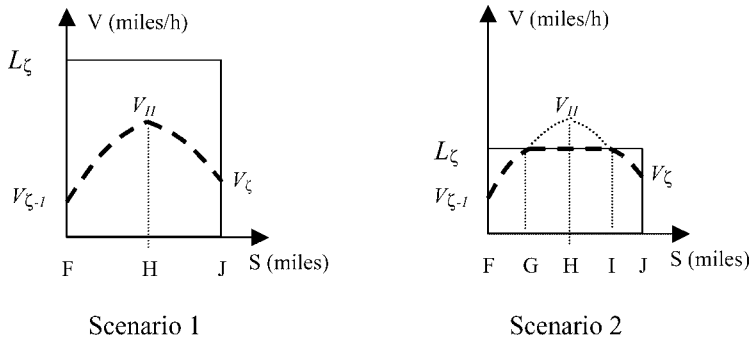

Fig. 5. Two different scenarios in step 3 of procedure Min_Travel_Time.

We first present the computation steps for determining the minimum travel time when a train travels a single fragment. Let the fragment be ζ. Given a train m as well as its acceleration rate a_1 and deceleration rate a_2, train m's velocity at the beginning of fragment ζ, $V_{\zeta-1}$, train m's velocity at the end of fragment ζ, V_ζ, the speed limit of fragment ζ, L_ζ, and the length of fragment ζ, S_ζ, we need to compute the minimal run time, t_ζ, for train m to travel fragment ζ without violating the speed limit.

Procedure Min_Travel_Time

Step 1. If $V_{\zeta-1} \le V_\zeta \le L_\zeta$ and $V_\zeta^2 - V_{\zeta-1}^2 = 2a_1 S_\zeta$, then continuously accelerate train m at rate a_1 to travel the length of fragment ζ. Return $t_\zeta = (V_\zeta - V_{\zeta-1})/a_1$ and stop. Otherwise, go to Step 2.

Step 2. If $V_\zeta \le V_{\zeta-1} \le L_\zeta$ and $V_{\zeta-1}^2 - V_\zeta^2 = 2a_2 S_\zeta$, then continuously decelerate train m at rate a_2 to travel the length of fragment ζ. Return $t_\zeta = (V_{\zeta-1} - V_\zeta)/a_2$ and stop. Otherwise, go to Step 3.

Step 3. If $V_{\zeta-1} \le L_\zeta$, $V_\zeta \le L_\zeta$, $V_\zeta^2 - V_{\zeta-1}^2 < 2a_1 S_\zeta$ and $V_{\zeta-1}^2 - V_\zeta^2 < 2a_2 S_\zeta$, then train m goes through an acceleration and deceleration process when traveling through fragment ζ. After the acceleration process, there may be some time where the train travels at a constant speed before starting its deceleration process. In scenario 1 in Figure 5, train m travels fragment ζ without ever traveling at a constant speed within the fragment. In scenario 2 in Figure 5, train m travels fragment ζ at a constant speed between the acceleration and deceleration processes. Let point H be the position that the acceleration process is changed to the deceleration process. We first calculate train m's velocity at point H, $V_H = \sqrt{\frac{a_1 V_\zeta^2 + a_2 V_{\zeta-1}^2 + 2a_1 a_2 S_\zeta}{a_1 + a_2}}$ (see the details in Appendix A).

If $V_H \le L_\zeta$ (Scenario 1 in Figure 5), return $t_\zeta = t_{FH} + t_{HJ} = \frac{V_H - V_{\zeta-1}}{a_1} + \frac{V_H - V_\zeta}{a_2} = -\frac{V_{\zeta-1}}{a_1} - \frac{V_\zeta}{a_2} + (\frac{1}{a_1} + \frac{1}{a_2})\sqrt{\frac{a_1 V_\zeta^2 + a_2 V_{\zeta-1}^2 + 2a_1 a_2 S_\zeta}{a_1 + a_2}}$.

If $V_H > L_\zeta$ (scenario 2 in Figure 5), return $t_\zeta = t_{FG} + t_{GI} + t_{IJ} = \frac{L_\zeta - V_{\zeta-1}}{a_1} + \frac{S_\zeta - (\frac{L_\zeta^2 - V_{\zeta-1}^2}{2a_1} + \frac{L_\zeta^2 - V_\zeta^2}{2a_2})}{L_\zeta} + \frac{L_\zeta - V_\zeta}{a_2}$.

Otherwise, go to Step 4.

Step 4. Train m cannot travel the fragment without violating the speed limit.

We next describe the computation of the minimum travel times when a train travels a distance that covers multiple fragments. This computation cannot be

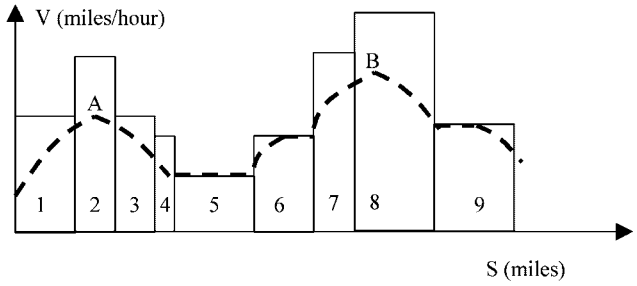

Fig. 6. An example of a train traveling in minimal time through multiple fragments.

determined by a single-pass procedure. We illustrate this point through an example. In Figure 6, we show the minimal run time (shown in dotted lines) of a train traveling through nine fragments. Points A and B in Figure 6 are two points where the train changes from accelerating to decelerating. To compute the smallest run time for a train traveling through multiple fragments, we have to determine these locations. If these points were determined in a forward sequence from fragment 1 to fragment 9 in the example in Figure 6, the train would start decelerating later than point A in fragment 2, which would generate an infeasible solution because in fragment 2 the train does not know that decelerating at any point later than point A would lead to a violation of the speed limit at fragment 5. Similarly, if these points were determined in a backward sequence from fragment 9 to fragment 1, the train would start decelerating at a point prior to point B in fragment 8, which would also generate an infeasible solution because in fragment 8 the train does not know that decelerating prior to point B would lead to a violation of the speed limit in fragment 6.

Next we give the formal representation of the problem to determine the smallest run time when a train travels through multiple fragments. Assume there are k fragments. Each fragment is S_i miles long and associated with a speed limit of $L_i, i = 1, 2, \ldots, k$. The total length of these k fragments is $S = \sum_{i=1}^{k} S_i$. Train m will travel these fragments in sequence. Train m's velocity at the beginning of fragment 1, V_0, and train m's velocity at the end of fragment k, V_k, are given. The objective is to determine t_i, train m's travel time at each fragment i, to minimize train m's total travel time $\sum_{i=1}^{k} t_i$ in distance S. Define a function $\Gamma(V_{\zeta-1}, V_\zeta, L_\zeta, S_\zeta)$ to represent the above Procedure Min_Travel_Time. The problem can be formalized as

$$
\begin{aligned}
&\mathit{Min} && \textstyle\sum_{i=1}^{k} t_i \\
&\text{such that} && \\
& && \Gamma(V_{i-1}, V_i, L_i, S_i) = t_i, \quad i = 1, 2, \ldots, k, \\
& && V_i \le \min(L_i, L_{i+1}), \qquad i = 1, 2, \ldots, k-1.
\end{aligned}
$$

The key issue in solving the above problem is to determine the velocity at the end of each fragment, $V_1, V_2, \ldots, V_{k-1}$. After determining these velocities, the optimal run time in each fragment can be computed by Procedure Min_Travel_Time. We refer to the algorithm that computes the velocities $V_1, V_2, \ldots, V_{k-1}$ as the Velocity_Augmenting algorithm.

Velocity_Augmenting algorithm

Step 0. Let Ω denote the set of unsolved subproblems. Each subproblem consists of one or more contiguous fragments and is defined by its beginning fragment, ending fragment, initial velocity, and final velocity. If $V_0 > L_1$ or $V_k > L_k$, the problem has no feasible solution. Otherwise, initially set $\Omega = \{(1, k, V_0, V_k)\}$.

Step 1. If Ω is null, stop. Return the obtained velocity at the end of each fragment. Otherwise, arbitrarily remove one subproblem (b, e, V_{b-1}, V_e) from Ω, where b and e are the indices of this subproblem's beginning fragment and ending fragment, respectively.

Step 2. Let $L'_{\min} = \min(L_b, L_{b+1}, \ldots, L_e)$. We denote by p the fragment with $L_p = L'_{\min}, b \le p \le e$. If the minimum is contained in multiple fragments, arbitrarily choose one of them as p. Call function $\Gamma(\min(L'_{\min}, V_{b-1}), \min(L'_{\min}, V_e), L'_{\min}, \sum_{i=b}^{e} S_i)$. If function Γ returns an infeasible solution, stop. The entire problem is infeasible. Otherwise, set train m's velocities at the beginning and end of fragment p, V_{p-1} and V_p, as the value indicated by the solution of function Γ.

Step 3. If fragment p is the only fragment in subproblem (b, e, V_{b-1}, V_e), the subproblem has been solved. Go to Step 1.

Step 4. If $p = b$, add subproblem $(b + 1, e, V_b, V_e)$ to Ω. Go to Step 1.

Step 5. If $p = e$, add subproblem $(b, e - 1, V_{b-1}, V_{e-1})$ to Ω. Go to Step 1.

Step 6. If $b < p < e$, add subproblem $(b, p - 1, V_{b-1}, V_{p-1})$ and subproblem $(p + 1, e, V_p, V_e)$ to Ω. Go to Step 1.

Step 0 of the algorithm initializes the problem. Step 1 finds a subproblem that contains a set of consecutive fragments and for which the velocities at the end of each fragment have not yet been determined. At step 2, Procedure Min_Travel_Time is called to solve the subproblem identified at Step 1 using the smallest speed limit of any fragment in the subproblem. Note that although the subproblem has multiple fragments, it is treated as a single fragment problem since a single speed limit (the smallest one) is used. From Procedure Min_Travel_Time, we can compute the speeds at the beginning and end of the fragment that has the smallest speed limit. These two speeds have to be included in any minimal travel time solution to the multiple fragment problem (see Proposition 3.1). Steps 3 to 6 eliminate the fragment with the smallest speed limit from the subproblem and regroup the remaining fragments into new subproblems.

In step 2, we use $\min(L'_{\min}, V_{b-1})$ and $\min(L'_{\min}, V_e)$ as the initial and final velocities of each subproblem, respectively. In Appendix B, we prove that the Velocity_Augmenting algorithm eventually delivers a solution that satisfies the boundary conditions that the train's initial and final velocities must be the specified values V_0 and V_k, respectively.

Using the same definitions, for each subproblem (b, e, V_{b-1}, V_e) in step 2 of the Velocity_Augmenting algorithm, we have the following proposition:

PROPOSITION 3.1. *If subproblem (b, e, V_{b-1}, V_e) has a feasible solution, then the velocities V_{p-1} and V_p obtained in step 2 of the Velocity_Augmenting algorithm are the optimal velocities at the beginning and end of fragment p for subproblem (b, e, V_{b-1}, V_e).*

PROOF. If $V_{p-1} = L'_{\min}$, the velocity V_{p-1} is the maximal speed that train m can be at the beginning of fragment p. Therefore, the velocity V_{p-1} is the

optimal velocity at the beginning of fragment p for subproblem (b, e, V_{b-1}, V_e). Otherwise, we have $V_{p-1} < L'_{\min}$ in the following two cases:

(1) Train m cannot reach the speed of $L'_{\min}$ at the beginning of fragment p even if train m continuously accelerates from the given velocity V_{b-1} at the beginning of fragment b.
(2) Train m will exceed the given velocity V_e at the end of fragment e even if train m continuously decelerates starting from a velocity $L'_{\min}$ at the beginning of fragment p.

For both of the above two cases, subproblem (b, e, V_{b-1}, V_e) cannot have a feasible solution with the speed at the beginning of fragment p greater than V_{p-1}. Therefore, the velocity V_{p-1} is the optimal velocity at the beginning of fragment p for subproblem (b, e, V_{b-1}, V_e). Using similar logic, we can prove that V_p is the optimal velocity at the end of fragment p for subproblem (b, e, V_{b-1}, V_e). □

PROPOSITION 3.2. *Let $V_0^*, V_1^*, \ldots, V_{k-1}^*, V_k^*$ be the optimal velocities at the beginning and end of each fragment for problem $(1, k, V_0, V_k)$. Let V_{p-1} and V_p be the two velocities at the beginning and end of fragment p obtained in step 2 of the Velocity_Augmenting algorithm for subproblem (b, e, V_{b-1}^*, V_e^*). Then, we have $V_{p-1}^* = V_{p-1}$ and $V_p^* = V_p$.*

Proposition 3.2 holds because the optimal solution of a subproblem becomes the global optimal if the given boundary conditions of the subproblem are the same as the conditions in the optimal context. In the Velocity_Augmenting algorithm, we first determine V_{p-1} and V_p of problem $(1, k, V_0, V_k)$. From Proposition 3.2, we have $V_{p-1} = V_{p-1}^*$ and $V_p = V_p^*$. Then, steps 4 to 6 of the Velocity_Augmenting algorithm guarantee the initial and final velocities for each new generated subproblem are both optimal velocities of the problem $(1, k, V_0, V_k)$. Therefore, the Velocity_Augmenting algorithm determines the optimal velocities for problem $(1, k, V_0, V_k)$.

In summary, computing the minimum run time for a train under multiple speed limits consists of two parts. First, the Velocity_Augmenting algorithm is called to determine the velocities at the end of each fragment. Second, the Procedure Min_Travel_Time is called to determine the smallest run time in each fragment. Since the complexity of Procedure Min_Travel_Time is a constant, $O(1)$, the total complexity of the second part is $O(k)$.

To consider the complexity of the first part, we need to consider the computational procedures in step 2 of the Velocity_Augmenting algorithm, which consists of the following four substeps:

(1) Find the minimal speed limit $L'_{\min}$ of the subproblem.
(2) Locate fragment p that has the speed limit of $L'_{\min}$.
(3) Call Procedure Min_Travel_Time to solve the subproblem under a single speed limit of $L'_{\min}$.
(4) Determine train m's velocities at the beginning and end of fragment p.

The complexity of substep 1 is $O[\log_2(k)]$; the complexity of substeps 2 and 3 is $O(1)$; the complexity of substep 4 is $O(k)$. Thus, the complexity of step 2

of the Velocity_Augmenting algorithm is $O(k)$. Since this step is performed at most k times, the complexity of the algorithm is $O(k^2)$

4. DEADLOCK-FREE ROUTING

Designing an efficient deadlock-free dispatching algorithm is a very important issue in both real systems and simulation models. Most current rail systems rely on the human dispatchers to route the trains. However, even for an expert human dispatcher, it is not easy to efficiently avoid deadlocks. To ensure no deadlock occurrence, the dispatcher may apply some very conservative dispatching rule or use some expert knowledge gained from experience. In this section, we propose a deadlock-free dispatching algorithm that routes the trains to their final destinations. Our algorithm is a heuristic since it does not guarantee to find a solution that minimizes the total trip times of the trains. We demonstrate the effectiveness of our algorithm and simulation methodology in the next section based on the actual system in Los Angeles County. Once it was shown that the current simulation methodology could effectively model train movement in complex rail networks, the model was used to test various expansion strategies of the system in order to accommodate anticipated increased demand.

When a train arrives at the stop checking point of a node, a routing algorithm is invoked to detect all the available successor nodes. If there are multiple available successor nodes, the node that gives the best performance for the system will be selected. The selection rules will be discussed at the end of this section. If there is no available successor node, the train will decelerate to prepare for stopping at the end of the node.

However, this flexible routing is very susceptible to deadlocks. A deadlock here means a situation that two or more trains cannot move anymore because each of them is requesting some resource held by other trains. A typical deadlock situation occurs when two trains running in the opposite direction are routed to nodes representing the same single-track line simultaneously. Typical methods to resolve deadlock are

(1) restore the previous status until no deadlock will happen,
(2) allow the preemption of the resource, and
(3) design some routing algorithm that avoids the deadlock (e.g., Banker's Algorithm, Habermann [1969]).

The first two methods do not work in our situation, because it is difficult and sometimes impossible for trains to move backward. Therefore, designing an efficient deadlock-free dispatching routing algorithm is necessary in our simulation modeling methodology.

Let $\Psi(T, \gamma(T, G))$ represent the state of the system, where T is the set of trains that are currently in network G and $\gamma(T, G) = \{(u, h, N_u) : u \in T, h \in G,$ and $N_u \subset G\}$ represents the relationship between T and G, where (u, h, N_u) signifies that train $u \in T$ currently has its head at node $h \in G$; and the train has also seized all the nodes in the node-set $N_u \subset G$ as well as the arcs and junctions associated with all the nodes in N_u. The state $\Psi(T, \gamma(T, G))$ can be

safe, unsafe, and deadlock. A safe state means there exists for each train in T a deadlock-free path to reach its destination. An unsafe state refers to the case where there is no current deadlock but a deadlock will occur in the future no matter which sequence of movements is taken. A deadlock state means that a deadlock has happened.

PROPOSITION 4.1. *Determining whether a state* $\Psi(T, \gamma(T, G))$ *is safe or unsafe is an NP-complete problem.*

A very similar problem exists in flexible manufacturing systems. For example, Lawley and Reveliotis [2001] proved that the safety problem for a SU-RAS (Single Unit-Resource Allocation System) is NP-complete. Their proof can be used to prove the above Proposition 4.1 with only a small modification.

An *ideal* deadlock avoidance algorithm should permit any train movement that results in a safe state and reject any train movement that only results in an unsafe state. The implication of Proposition 4.1 is that it is not known whether or not there exists an ideal deadlock avoidance algorithm that runs in polynomial time. Note that an *optimal* routing algorithm that minimizes the total train delay time must be able to do ideal deadlock avoidance. Therefore, it is thought to be computationally prohibitive in the worst case to determine the optimal routes that minimize total train delay time in a real-time environment.

Although developing an *optimal* routing algorithm is thought to be computationally prohibitive in the worst case, there exist polynomial-time deadlock-free routing algorithms. Any algorithm that permits any train movement that results in a subset of the safe states set is deadlock-free. But these deadlock-free algorithms may not be efficient enough in dispatching. For example, the subset of the safe states set considered in the algorithm may be too small, resulting in too many safe states being treated as unsafe states. Algorithm 4.1 is such a simple deadlock-free routing algorithm. Before presenting Algorithm 4.1, we define the concept of *free path*. A free path means that the resources of all the nodes and arcs in this path, except the start node and end node, are free. Algorithm 4.1 avoids deadlock by only allowing a train to move to a successor node if it is on a free path to the train's destination and that destination has available capacity.

Algorithm 4.1. Arbitrarily choose a node that satisfies the following two conditions from the successor node candidate list as the selected successor node.

(1) It is on a free path from the node in which the train's head is currently located to the train's destination node.
(2) The train's destination node has available capacity.

If no such node exists, decelerate the train immediately.

PROPOSITION 4.2. *Algorithm* 4.1 *is deadlock-free.*

By definition, a free path cannot lead to the selection of a successor node that ultimately results in a deadlock. Therefore Algorithm 4.1 cannot possibly result in a deadlock. Although Algorithm 4.1 is simple, it is a very conservative deadlock-free dispatching strategy and the performance when applying it to a rail network can be poor. For instance, assume there is only a single-track

line connecting stations A and B. Ten trains need to depart from station A to station B. The ready times for these trains at station A are 0, 1, 2, ..., 9, respectively. The travel times of all these trains from station A to station B are the same, 10 hours. The most efficient way to dispatch these 10 trains is to dispatch them right at their ready time, assuming 1 hour is a sufficient headway between the trains. After 19 hours, the last train will arrive at station B. But using Algorithm 4.1, the hours elapsed before the last train arrives at station B increases to 100.

We present Algorithm 4.2 to improve the performance of Algorithm 4.1. Before presenting this algorithm, we make some definitions. A *buffer* between node i and node j is a set of nodes connected as a chain between node i and node j. The length of the buffer is the sum of the lengths of all the nodes in this buffer. Algorithm 4.2 is still a one-step look-ahead algorithm. However, the main difference between Algorithms 4.1 and 4.2 is that in the latter algorithm a train can still be dispatched through a successor node j as long as there is an available buffer that passes through node j. This extra buffer guarantees that the train does not have to stop at node j if it is unsafe.

Before presenting the algorithm, we describe Procedure Move_Forward_Check, which determines whether train m can move to a successor node j or through node j as part of a chain of nodes (referred to as a *buffer*). The procedure returns either **safe *j*** (move to successor node j), **safe buffer *b**** (move through the nodes including node j in the chain in buffer b^*), or **unsafe** (which means a move to node j may lead to a deadlock). Assume that train m with its head currently at node i is being evaluated. The current state of the system is $\Psi(T, \gamma^0(T, G))$.

Procedure Move_Forward_Check

Step 0. Let $\gamma^1(T, G)$ be the relationship after the movement of train m to node j from $\gamma^0(T, G)$. Let the resulting state be $\Psi(T, \gamma^1(T, G))$.

Step 1. Iteratively remove each train q from T, where there is at least one free path to train q's destination and train q's destination has available capacity. Let T^1 be the trains in T that were not removed. Let $\gamma^2(T^1, G)$ and $\Psi(T^1, \gamma^2(T^1, G))$ be the resulting relationship and state with only trains T^1 in the network, assuming that all the trains in $\neg T^1 \equiv T - T^1$ have completed their required movements and have departed the network, freeing all their associated resources.

Step 2. If train $m \notin T^1$, it is safe to move train m forward. Go to step 7.

Step 3. If train $m \in T^1$, for each path from node j to train m's destination, find a buffer if it exists between node j and the first occupied node on the path in relationship $\gamma^2(T^1, G)$. Place all identified buffers in set B.

Step 4. If B is null, it is unsafe to move train m forward. Go to step 9. Otherwise, remove a buffer b^* from B. Let $\gamma^3(T^1, G)$ and $\Psi(T^1, \gamma^3(T^1, G))$ be the resulting relationship and state after the movement of train m to the end of buffer b^* from relationship $\gamma^2(T^1, G)$.

Step 5. Iteratively remove each train q from T^1, where there is at least one free path to train q's destination and train q's destination has available capacity. Let T^2 be the trains in T^1 that were not removed. Let $\gamma^4(T^2, G)$ and $\Psi(T^2, \gamma^4(T^2, G))$ be the resulting relationship and state with only trains T^2 in the relationship $\gamma^3(T^1, G)$.

Step 6. If train $m \notin T^2$, it is safe to move train m forward. Go to step 8. Otherwise, Go to step 4 and try another buffer in B.

Step 7. It is safe to move train m from node i to node j, return **safe** $\boldsymbol{j}$, and stop.
Step 8. It is safe to move train m from node i to the end of buffer b^*, return **safe buffer** $\boldsymbol{b}^*$, and stop.
Step 9. It is unsafe to move train m from node i to node j, return **unsafe**, and stop.

Note that set T^1 consists of all the trains in T for which there is no free path to the train's destination or the train's destination has insufficient available capacity in state $\Psi(T^1, \gamma^2(T^1, G))$ at step 1. Nevertheless in state $\Psi(T^1, \gamma^3(T^1, G))$ at the end of step 4, some of the trains in set T^1 may have a free path to their destinations for the following reasons: after we move train m forward to the end of its buffer, the resources that are released by completion of this movement may enable some trains in set T^1 to move to their destinations. Next, we give a formal statement of Algorithm 4.2.

Algorithm 4.2.

Step 0. Assume that the current system state is $\Psi(T, \gamma^0(T, G))$ and train set $T^* = T$.
Step 1. Arbitrarily remove a train q from set T^*.
Step 2. Call Procedure Move_Forward_Check for train q at system state $\Psi(T, \gamma^0(T, G))$. If Procedure Move_Forward_Check returns **safe** $\boldsymbol{j}$, train q immediately seizes all resources of node j and the system is updated to a new state $\Psi(T, \gamma^1(T, G))$, and stop Algorithm 4.2. If procedure Move_Forward_Check returns **safe buffer** $\boldsymbol{b}^*$, train q seizes all the resources of node j and the nodes in buffer b^* and the system is updated to a new state $\Psi(T, \gamma^3(T, G))$, and stop Algorithm 4.2. If Procedure Move_Forward_Check returns **unsafe**, go to step 1.

Note that there exists at least one train that can move forward at any system state as long as Algorithm 4.2 is deadlock-free. Therefore, there will exist a train q in which step 2 of Algorithm 4.2 will return either **safe** $\boldsymbol{j}$ or **safe buffer** $\boldsymbol{b}^*$. Next, we prove that Algorithm 4.2 is deadlock-free.

PROPOSITION 4.3. *Algorithm* 4.2 *is deadlock-free.*

PROOF. Assume that state $\Psi(T, \gamma^0(T, G))$ is the current system state and $\Psi(T, \gamma^0(T, G))$ is safe. Furthermore, if we apply Algorithm 4.2 in state $\Psi(T, \gamma^0(T, G))$, the system state changes to state $\Psi(T, \gamma^1(T, G))$ by moving train m from node i to node j or the system state changes to state $\Psi(T, \gamma^3(T, G))$ by moving train m from node i to the end of buffer b^*. To prove Algorithm 4.2 is deadlock-free, we need to prove that both states $\Psi(T, \gamma^1(T, G))$ and $\Psi(T, \gamma^3(T, G))$ are safe states.

First, we assume that calling Procedure Move_Forward_Check at step 2 of Algorithm 4.2 for train m in state $\Psi(T, \gamma^0(T, G))$ returns **safe** $\boldsymbol{j}$ so that $m \in \neg T^1 \equiv T - T^1$; and we prove that, in this case, $\Psi(T, \gamma^1(T, G))$ is a safe state.

In Algorithm 4.2, apply Procedure Move_Forward_Check to check whether it is possible to move train *m's head* from node i to node j in state $\Psi(T, \gamma^0(T, G))$. Then, state $\Psi(T, \gamma^1(T, G))$ is the output state of Step 0 and state $\Psi(T^1, \gamma^2(T^1, G))$ is the output state of step 1. Let $\gamma^0(T, G) = \{(u, h^0, N_u^0) : u \neq m, u \in T, h^0 \in G, \text{ and } N_u^0 \subset G\} \cup \{(m, i, N_m^0)\} = \{(u, h^0, N_u^0) : u \neq m, u \in T^1\} \cup \{(u, h^0, N_u^0) : u \neq m, u \in \neg T^1\} \cup \{(m, i, N_m^0)\}$. Define $\gamma^c(T^1, G) = \{(u, h^0, N_u^0) : u \neq m, u \in T^1\}$. Since Algorithm 4.2 returns **safe** $\boldsymbol{j}$, we have

$m \in {}^{\neg}T^1, \gamma^c(T^1, G) = \{(u, h^0, N_u^0) : u \in T^1\}$. Next, we show that state $\Psi(T^1, \gamma^c(T^1, G))$ is a safe state by contradiction. If state $\Psi(T^1, \gamma^c(T^1, G))$ is *unsafe*, there exists a set of trains $q_1, q_2, \ldots, q_s$ that form a deadlock now or in the future. Since $\gamma^c(T^1, G) \subset \gamma^0(T, G)$, in state $\Psi(T, \gamma^0(T, G))$ and state $\Psi(T^1, \gamma^c(T^1, G))$, trains $q_1, q_2, \ldots, q_s$ seize the same resources and their heads are in the same nodes. Therefore, the same deadlock will happen in state $\Psi(T, \gamma^0(T, G))$. This is a contradiction to the assumption that state $\Psi(T, \gamma^0(T, G))$ is safe. Hence, state $\Psi(T^1, \gamma^c(T^1, G))$ is safe.

Next, we prove that $\gamma^2(T^1, G) = \gamma^c(T^1, G)$, where $\gamma^2(T^1, G)$ is the resulting relationship of step 1. First, we have $\gamma^1(T, G) = \{(u, h^0, N_u^0) : u \in T^1\} \cup \{(u, h^0, N_u^0) : u \neq m, u \in {}^{\neg}T^1\} \cup \{(m, j, N_m^1)\}$, where N_m^1 is the set of all nodes that have been seized and allocated to train m after train m has been moved to node j. Note that in step 1 we only remove the trains in set ${}^{\neg}T^1$ including train m and do not change any train's position in train set T^1. Therefore, $\gamma^2(T^1, G) = \{(u, h^0, N_u^0) : u \in T^1\} = \gamma^c(T^1, G)$ and state $\Psi(T^1, \gamma^2(T^1, G)) = \Psi(T^1, \gamma^c(T^1, G))$ is a safe state.

Thus, state $\Psi(T, \gamma^1(T, G))$ is safe because it can be reduced to a safe state $\Psi(T^1, \gamma^2(T^1, G))$ by a finite number of operations by moving all the trains in train set ${}^{\neg}T^1$ including train m to their destinations in state $\Psi(T, \gamma^1(T, G))$.

We next consider the case in which Procedure Move_Forward_Check returns **safe buffer** $\boldsymbol{b}^*$ such that $m \in T^1$ in step 1 but $m \notin T^2$ in step 6; and in this situation, we prove that $\Psi(T, \gamma^3(T, G))$ is a safe state. The proof is similar to the above argument with the following substitutions: replace T with T^1, replace γ^0 with γ^2, replace the initial safe state $\Psi(T, \gamma^0(T, G))$ with a safe state $\Psi(T^1, \gamma^2(T^1, G))$, replace T^1 with T^2, replace γ^2 with γ^4, and replace γ^1 with γ^3. Using similar logic in proving state $\Psi(T^1, \gamma^2(T^1, G))$ is safe, we can prove that state $\Psi(T^2, \gamma^4(T^2, G))$ is safe. State $\Psi(T, \gamma^3(T, G))$ is safe because it can be reduced to a safe state $\Psi(T^2, \gamma^4(T^2, G))$ by a finite number of operations by moving all the trains in train set ${}^{\neg}T^2$ including train m to their destinations in state $\Psi(T, \gamma^3(T, G))$. □

Assume that at state $\Psi(T, \gamma^0(T, G))$, $|T| = q$ and G has n nodes and a arcs. Step 1 of Procedure Move_Forward_Check can be completed through a breadth-first-search on graph G. This step is of complexity $O(n + a)$ at most. The step of determining all the buffers for a train is of complexity $O(qn^2)$ at most. A train can have no more than n buffers. For each buffer, step 5 is performed with the same complexity as step 2. Therefore, the worst-case computational complexity of the Procedure Move_Forward_Check is of $O(qn^2 + na)$. Thus, the worst-case computational complexity of Algorithm 4.2 is $O(qn^3 + n^2a)$ since the algorithm is evaluated for each successor node.

From an efficiency point of view, Algorithm 4.2 is much better than Algorithm 4.1. For example, Algorithm 4.2 will provide the optimal dispatching times to the single-track two-station example given above. The example in Figure 7 shows a situation where Algorithm 4.2 gives the optimal dispatching times while Algorithm 4.1 gives a poor solution. Assume there are two trains, trains m_1 and m_2. They are ready for departure at nodes 2 and 1, respectively, at time 0. Their destination nodes are nodes 1 and node 2, respectively. As a result, the

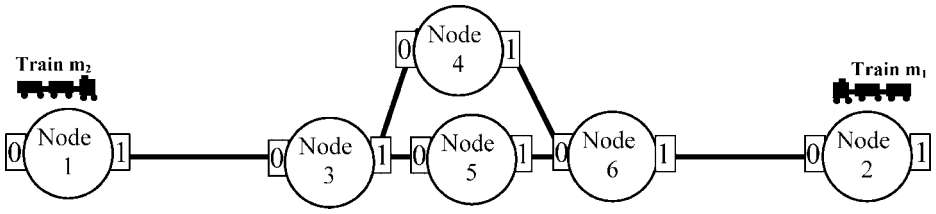

Fig. 7. An example that Algorithm 4.2 can solve efficiently.

movement of these two trains using Algorithm 4.2 will be as follows:

(1) Train m_1 leaves node 2 immediately, seizes the resources of node 6, and moves to node 6.
(2) After train m_1 leaves node 2, train 2 can be introduced into the network, because buffer node 5 exists. Train m_2 seizes all the resources of node 3 and node 5 and moves to node 5.
(3) Once train m_1 arrives at node 6, it considers node 4 as a buffer between itself and train m_2. Therefore, it seizes all the resources of node 4 and moves to node 4.
(4) When train m_1 arrives at node 4 and train 2 arrives at node 5, they will go to their respective destinations without stopping.

By performing the above dispatching, both trains can move in opposite directions simultaneously. Furthermore, if all node lengths are equal and greater than the train lengths, we can guarantee no train will stop in this example using Algorithm 4.2. Note that applying Algorithm 4.1 to this example would needlessly add waiting times. Algorithm 4.1 would force train m_2 to wait until train m_1 arrives at node 1 before releasing it to the network.

The algorithms presented above arbitrarily break ties when there are multiple available successor nodes. We next present an improved method for selecting the best available successor node. The method accounts for the following three factors:

(1) the maximum priority difference between the current train and the immediate successor train running in the same direction if one exists,
(2) the maximal number of trains running in the same direction along the path from the successor node to the train's destination node, and
(3) the minimum travel time for the current train from the successor node to the current train's destination node assuming there is no downstream conflicting traffic ahead of the current train.

Each above factor emphasizes a different aspect of operating efficiency. If priority is based on the speed of the train, the first condition allows higher-speed trains to bypass slower-speed trains. The second condition maximizes the number of trains moving in the same direction for a given path, thus freeing other paths for opposing-moving trains. The third condition minimizes the travel time for the train. In our system, we consider the above factors in sequential order to break ties.

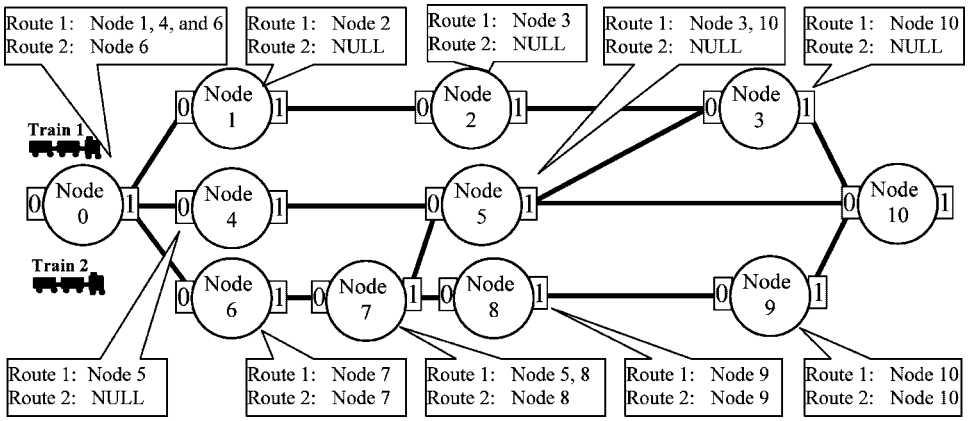

Fig. 8. A routing table example for different routes in a triple-track system.

5. MODEL INTEGRATION AND VALIDATION

In this section, we first describe how to integrate the concepts and methodologies outlined in Sections 2 to 4 to model multiple track rail systems including train acceleration and deceleration. Then we demonstrate the effectiveness of using the proposed algorithms in Sections 3 and 4 in simulating train movements in the rail network from Downtown Los Angeles to the Inland Empire Trade Corridor.

5.1 Integrated Model

The integrated simulation model consists of five parts: preprocessing, event calendar, event processing mechanics, Velocity_Augmenting algorithm, and central dispatching algorithm. We discussed the logic associated with the event calendar and the different event types in Section 2.2. The Velocity_Augmenting algorithm and central dispatching algorithm were described in Sections 3 and 4, respectively. In this section, we will mainly focus on preprocessing and the integration of these five independent parts.

The objective of preprocessing is to generate the routing table associated with each node. The routing table at each node stores the possible successor nodes list for each route at this node. By designing a proper routing table at each node, we can specify the train's movement in the physical network. For instance, in the triple-track system in Figure 8, there are two trains both departing from node 0 and ending at node 10. Train 1 is a passenger train and train 2 is a freight train. Because the speed limit of the freight train is lower than the speed limit of the passenger train, we force the freight trains to always take the right-most-track in the direction of their movement to reduce the probability that the slow-moving train blocks the fast-moving trains like passenger trains. Therefore, these two train types have two different possible path sets, although they have the same departure and destination nodes. Let us define route 1 for train 1 and route 2 for train 2. Figure 8 shows the contents of the routing table at each node for these two routes. For each node, we provide the successor

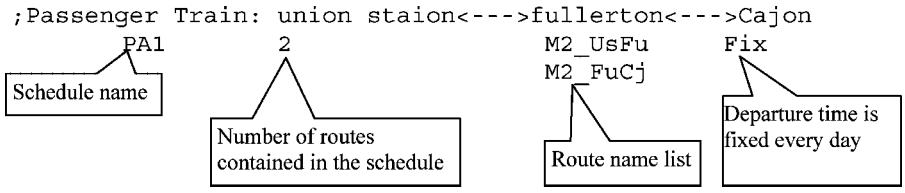

Fig. 9. A sample schedule data file.

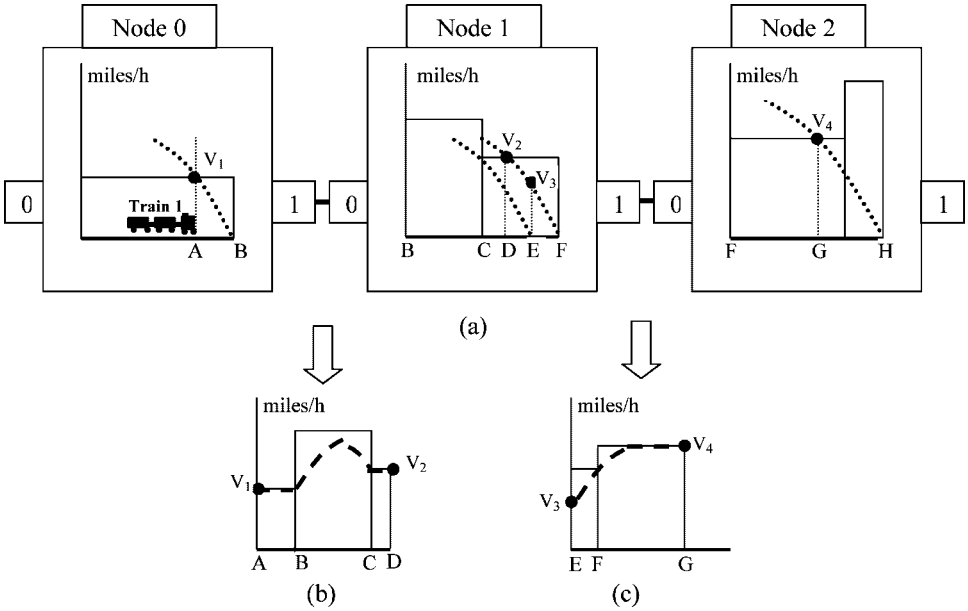

Fig. 10. Illustration of train movement.

node in the routing table for the two routes. The routing table at each node is generated based on a depth-first-search-like algorithm.

The train entities in the simulation system are created based on the schedule information read from a data file in the preprocessing stage. Each schedule contains information on the train type, list of routes, and the departure time of each train every day. In Figure 9, we give an instance of the schedule data file. Note that each schedule may have multiple routes. This is used to represent the schedule of passenger trains, because for passenger trains there may be intermediate stations that must be visited. Each route starts from one intermediate station and ends at the next intermediate station.

After a train entity is created at the scheduled departure time, it enters the system waiting for the resources identified in the departure node. Once seized, the train runs at the speed computed through the Velocity_Augmenting algorithm and controlled based on the central dispatching algorithm. In Figure 10 we use an example to illustrate the train movement in the simulation system. Assume that train 1's head is at node 0's stop checking point, point A, at a current velocity of V_1. Train 1 sends a request to the central dispatching algorithm

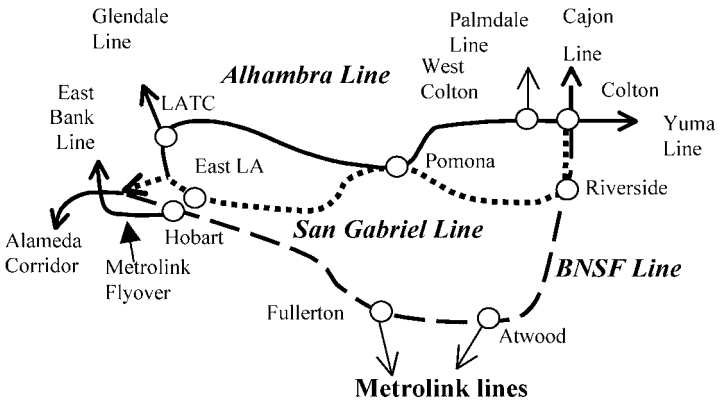

Fig. 11. Rail network of the Los Angeles-Inland Empire Trade Corridor.

to ask for the next operation. Suppose that the central dispatching algorithm finds that train 1 can enter node 1 without causing any potential deadlock. Train 1 first calculates the position of the stop checking point at node 1, point D, as well as the speed V_2 at point D. Then the Velocity_Augmenting algorithm is called to compute the minimum time to move train 1 from point A to point D while observing all the intermediate speed limits with a start velocity of V_1 and an end velocity of V_2, as shown in Figure 10(b).

After train 1's head arrives at point D, suppose the central dispatching algorithm finds that train 1 cannot enter node 2 due to the unavailability of a resource identified in node 2. Train 1 begins to decelerate immediately at point D. If the resources do not become available before the train reaches point F, the train will come to a full stop at point F. Based on the definition of the stop checking points, the train will be able to come to a full stop at point F if it decelerates at the given rate. In reality, some mandatory buffer space may be required between trains. This buffer space can be used as a safe headway between running trains or to make sure the trains do not block a junction when stopping. To account for this issue in our model, we only need to force the full stop point of a train to be at some point before the end of the node instead of right at the end of the node if a stop is necessary.

If the central dispatching algorithm finds that train 1 can enter node 2 when train 1 is in its deceleration process with its head at point E, the model calculates the position of point G, the stop checking point at node 2, as well as the speed V_4 at point G. Then the Velocity_Augmenting algorithm is called to compute the minimum travel time to move from point E to point G, as shown in Figure 10(c).

5.2 Model Validation and Simulation Results

The above simulation modeling methodology was applied to the rail network from Downtown Los Angeles to the Eastern Inland Empire area. Figure 11 shows the rail network in this area.

The purpose of the study was to analyze the capacity of the rail network in this area. Currently, there are 195 miles of track. We divided this track into 330 segment resources and 178 junction resources. From these resources,

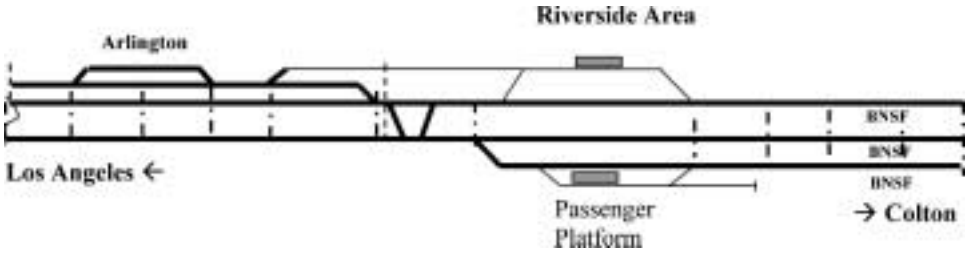

Fig. 12. A portion of the rail network near Riverside station decomposed into track segments.

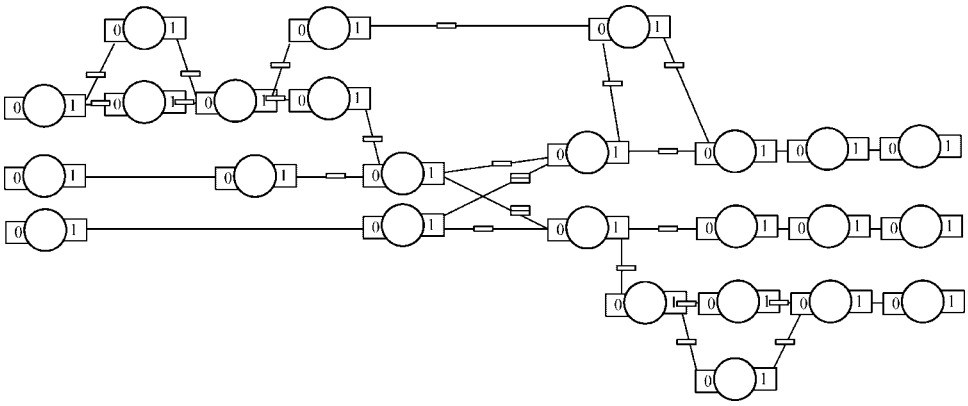

Fig. 13. Simulation network for the track segments shown in Figure 12.

we developed a network architecture consisting of 412 nodes and 593 arcs. Figure 12 shows a *small* portion of the network decomposed into the track segments and Figure 13 shows the resulting node and arc network (for simplicity we have omitted the text information within the nodes).

Today, there are close to 141.5 freight trains per day and 101 passenger trains per day that use this portion of the rail network. As forecast by the Los Angeles Economic Development Corporation, these numbers are expected to increase to 278.2 freight trains per day and 227 passenger trains per day by 2025. Using our simulation modeling methodology, we identified bottleneck points where additional trackage will be required to meet the increased demand (see Dessouky et al. [2002] for a discussion on the details of the capacity expansion plans).

Before the detailed capacity expansion study was performed, we first demonstrated the effectiveness and validity of our simulation methodology, especially the Velocity_Augmenting algorithm presented in Section 3 and the central dispatching algorithm presented in Section 4. We compared our approach (case 4) with three other approaches (cases 1, 2, and 3) using simple speed control and dispatching strategies under the current trackage configuration and train frequencies. The studied approaches were as follows:

—Case 1 uses a simple speed control strategy, which moves the train at a velocity equal to the lowest speed limit, with Algorithm 4.1 as the dispatching strategy.

Table I. Results of Four Different Cases of Freight Trains

	Case 1	Case 2	Case 3	Case 4
Average freight train flow time (min)	205	147	127	117
Average freight train delay time (min)	115	60	37	30
Standard error	6.5	1.3	0.2	0.1

Table II. Results of Four Different Cases of Passenger Trains

	Case 1	Case 2	Case 3	Case 4
Average Amtrak passenger train flow time (min)	146	66	55	49
Amtrak standard error	2.3	0.7	0.1	0.1
Average Metrolink passenger train flow time (min)	160	100	60	57
Metrolink standard error	2.3	0.8	0.1	0.1

—Case 2 uses the Velocity_Augmenting algorithm to control the speed, with Algorithm 4.1 as the dispatching strategy.

—Case 3 uses the simple speed control strategy, with Algorithm 4.2 as the dispatching strategy.

—Case 4 uses the Velocity_Augmenting algorithm to control the speed, with Algorithm 4.2 as the dispatching strategy.

Ten independent replications were made for each case with each run consisting of 100 days, and the statistics are cleared after 10 simulated days. Each run took around 20 CPU minutes on a Pentium III 1-GHZ processor. The results are shown in Tables I and II. The results for freight trains are shown in Table I while the results for passenger trains, Amtrak and Metrolink, are shown in Table II. The flow time is defined to be the total time a train is in the system, and the delay time is the difference between the simulation flow time and the theoretical fastest flow time. The theoretical fastest flow time is set as the shortest travel time between the departure and destination nodes assuming there is no downstream conflicting traffic ahead of the train. It is determined by making a single run for each train and route combination in the schedule. Since the delay time equals the flow time minus a constant, the standard errors of the flow time and delay time are the same.

Comparing the results of case 1 with case 4 in Tables I and II, we can conclude that using the algorithms introduced in Sections 3 and 4 to control the speed and dispatch the trains can dramatically reduce the average flow time and average delay. Furthermore, applying these algorithms also makes the system more stable, because the standard error is significantly reduced. This shows the benefit of using both the complex speed control and dispatch algorithms together. These results were shown to representatives of the Southern California Association of Governments, and they commented that current delays experienced by the trains most closely match the results of case 4.

6. CONCLUSION

We developed a deadlock-free simulation methodology to model rail networks that consist of a multiple trackage configurations and speed limits. The modeling methodology can be used to simulate any type of rail network consisting of any number of tracks. The modeling methodology is based on a general graphical technique which can easily represent simple and complex rail networks.

The model also determines the fastest the train can travel at any instant of time. When the acceleration and deceleration rates are finite, we showed that the fastest the train can travel at each instant of time is a complex combination of the acceleration and deceleration rates and the uniform motion of the train. Our algorithm computes the train travel time in polynomial time as a function of the number of constraining speed limits (i.e., the number of fragments).

We implemented our simulation methodology to model the movement of passenger and freight trains in Los Angeles County, and the model was used to identify the needed capacity requirements to meet anticipated increases in freight rail traffic in the region.

APPENDIX A

To derive the formula of computing V_H for the two scenarios in Figure 5, consider the following three equations:

$$V_H^2 - V_{\zeta-1}^2 = 2a_1 S_{FH},$$
$$V_H^2 - V_\zeta^2 = 2a_2 S_{HJ},$$
$$S_{FH} + S_{HJ} = S_\zeta.$$

Eliminate S_{FH} and S_{HJ} in the above equations to arrive at the formula for V_H: $V_H = \sqrt{\frac{a_1 V_\zeta^2 + a_2 V_{\zeta-1}^2 + 2a_1 a_2 S_\zeta}{a_1 + a_2}}$.

APPENDIX B

Given k contiguous fragments, each fragment i is associated with a speed limit of L_i. Train m travels these fragments in sequence. Train m's velocity at the beginning of fragment 1, V_0, and train m's velocity at the end of fragment k, V_k, are given. The objective is to determine the smallest run time for train m to travel all these fragments. Next we prove that if the Velocity_Augmenting algorithm described in Section 3 can find an optimal solution for this problem, this optimal solution must satisfy the boundary conditions that train m's initial and final velocities must be the specified values V_0 and V_k, respectively.

PROOF. First, note that $V_0 \leq L_1$, where L_1 is the speed limit of fragment 1; otherwise, the problem is infeasible. Consider all the subproblems that include fragment 1 and are created and solved in the Velocity_Augmenting algorithm. Assume that the speed limit of subproblem $(1, e, V_0, V_e)$ is $L'_{\min}$, where $L'_{\min}$ is the largest minimal speed limit of all the subproblems that contain fragment 1. Then, we have $L'_{\min} = L_1$. First, if $L'_{\min} < L_1$, according to the Velocity_Augmenting algorithm, a new subproblem $(1, k, V_0, V_k)$ will be generated

with speed limits greater than $L'_{\min}$, where fragment k is the fragment immediately before the fragment with speed limit of $L'_{\min}$ in subproblem $(1, e, V_0, V_e)$. This is a contradiction with the assumption that $L'_{\min}$ is the largest minimal speed limit of all the subproblems that contain fragment 1. Second, since subproblem $(1, e, V_0, V_e)$ contains fragment 1, we have $L'_{\min} \leq L_1$. Therefore, $L'_{\min}$ must equal L_1.

According to the Velocity_Augmenting algorithm, the velocities at the beginning and end of fragment 1 in the solution of subproblem $(1, e, V_0, V_e)$ equal the velocities at the beginning and end of fragment 1 in the optimal solution of the entire problem. Because $L'_{\min} = L_1$ and $L_1 \geq V_0$, the initial velocity of subproblem $(1, e, V_0, V_e)$ equals V_0. Therefore, the initial velocity of the optimal solution equals V_0. Using similar logic, we can prove that the final velocity of the optimal solution must equal V_k. □

ACKNOWLEDGMENTS

We also appreciate the Area Editor and three anonymous referees for their valuable comments.

REFERENCES

BUECHE, F., HECHT, E., AND BUECHE, F. J. 1997. *Schaum's Outline of College Physics*. McGraw-Hill, New York, NY.

CAREY, M. 1994. A model, algorithm and strategy for train pathing, with choice of lines, platforms and routes. *Trans. Res. 28(B)*, 333–353.

CHENG, Y. 1998. Hybrid simulation for resolving resource conflicts in train traffic rescheduling. *Comput., Indust. 35*, 3, 233–246.

DESSOUKY, M. M. AND LEACHMAN, R. C. 1995. A simulation modeling methodology for analyzing large complex rail networks. *Simulation 65*, 2, 131–142.

DESSOUKY, M. M., LEACHMAN, R. C., AND LU, Q. 2002. Using simulation modeling to assess rail track infrastructure in densely trafficked metropolitan areas. In *Proceedings of the 2002 Winter Simulation Conference* (San Diego, CA, Dec.), E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Carnes, Eds. Winter Simulation Conference, Piscataway, NJ.

HABERMANN, A. N. 1969. Prevention of system deadlocks. *Commun. ACM 12*, 373–377.

HIGGINS, A. AND KOZAN, E. 1998. Modeling train delays in urban networks. *Transport. Sci. 32*, 4, 346–357.

KOMAYA, K. 1991. A new simulation method and its application to knowledge-based systems for railway scheduling. In *Proceedings of the IEEE/ASME Joint Railway Conference* (St. Louis, MO, May). IEEE/ASME, New York, NY, 59–66.

LAWLEY, M. A. AND REVELIOTIS, S. A. 2001. Deadlock avoidance for sequential resource allocation systems: Hard and easy cases. *Internat. J. FMS 13*, 4, 385–404.

LEACHMAN, R. C. 1991. Railroad capacity and operations analysis for the Alameda Consolidated Transportation Corridor Project. Tech. rep. Leachman & Associates, Berkeley, CA.

LEWELLEN, M. AND TUMAY, K. 1998. Networks simulation of a major railroad. In *Proceedings of the 1998 Winter Simulation Conference* (Washington, DC, Dec.), D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, Eds. Winter Simulation Conference, Piscataway, NJ, 1135–1138.

PETERSEN, E. R. AND TAYLOR, A. J. 1982. A structured model for rail line simulation and optimization. *Transport. Sci. 16*, 2, 192–206.

VICKERMAN, M. J. 1998. Next-generation container vessels: Impact on transportation infrastructure an operation. *TR News* 196, 3–15.

Received July 2002; revised January 2003, May 2003, August 2003, November 2003; accepted November 2003