2017 American Control Conference
Sheraton Seattle Hotel
May 24–26, 2017, Seattle, USA

Scheduling of Vehicle Movement in Resource-Constrained Transportation Networks using a Capacity-Aware Heuristic

Harshad Khadilkar[1]

Abstract— **This paper proposes a novel vehicle movement scheduling heuristic for transportation networks with controlled access and limited infrastructure resources at each node and edge. The scheduling problem is known to be computationally NP-hard, making exact solutions difficult to compute. Several prior studies have described and analysed a class of approximate scheduling methods known as *travel advance heuristics* (TAH), where vehicles are moved forward on their planned routes one node at a time. One move implies allotting the vehicle a resource at the current node and the subsequent edge for a fixed period of time. The order in which such moves are made is important for avoiding deadlock (ensuring that the desired move is feasible) as well as for ensuring good schedule quality (small travel times). In this paper, a version of TAH designated as *critical-first* (TAH-CF) is shown to be provably deadlock-free under certain conditions on the initial state of the network. This guarantee is provided without requiring any future move exploration as mandated by previously published methods. In addition to deadlock avoidance, TAH-CF is also shown to compute schedules with smaller travel times than those produced by a fixed priority version (TAH-FP). Two large, realistic test cases in the railway domain demonstrate the advantages of the proposed method.**

I. INTRODUCTION

Transportation networks with controlled access are those networks where vehicular access and movement is governed by an *operator* rather than by the vehicles themselves. Examples of such networks are railways and commercial air transportation, where the job of the operator is performed by rail dispatchers and air traffic controllers respectively. By contrast, public road networks have (largely) uncontrolled access, where vehicles can autonomously choose their routes and times of movement. In controlled access networks, desired routes and movement times for each vehicle are declared in advance. The operator's job is to allot resources to each vehicle on its desired route, typically for exclusive use by the vehicle for a fixed period of time. Any resulting resource conflicts between vehicles are resolved by deviating from the desired times (assigning delays) and less frequently, by changing routes (diversion). The computation of resource allocations and times can be formulated as an optimisation problem. The mathematical structure is analogous to job shop scheduling [1], and has been shown to be NP-hard [2].

In large networks over long time horizons, finding exact solutions can thus require an exponential number of computations. Consequently, approximate optimisation algorithms [3], [4] have attracted interest in the job shop as well as transportation literature. Popular approaches include decomposition into subnetworks [5] and heuristic algorithms [6]. However, a major problem with approximate methods is that of deadlock situations, where vehicles are assigned resources such that their desired moves become infeasible. These situations are a result of myopic scheduling, where local decisions have unexpected global effects. If possible, deadlocks are resolved by backtracking: partially rolling back previous allocations and changing the decisions in some predefined manner. Alternatively, some authors have proposed scheduling strategies that are inherently resistant to deadlocks [7], [8]. Such *deadlock avoidance* strategies typically involve checking certain conditions for onset of deadlocks [9], [10], [11], thus requiring more computational steps than basic heuristic methods. Previous approaches that focus on regulating rate of traffic input into the system [12], [13], [14] tend to reduce travel times (improve schedule quality), but are not guaranteed to prevent deadlocks.

By contrast, the scheduling algorithm presented in this paper provides deadlock avoidance, has minimal computational overhead, and produces good quality schedules. The approach falls under the broad category of travel advance heuristics (TAH) [15], [16], and is described in detail in Section II. The novel feature introduced in this work is the order in which vehicle movements are chosen, which is computed using a heuristic called *critical-first* (TAH-CF). Section III shows that under certain conditions governing the initial state of the system, TAH-CF is provably deadlock/backtracking-free. As a consequence of this property, each vehicle movement is performed in an environment with a low number of prior resource allocations (constraints), thus reducing the probability of resource conflicts. Since resource conflicts are resolved by delaying one or more vehicles, fewer conflicts are expected to result in lower levels of unscheduled delay. The heuristic is implemented on several realistic railway network test cases in Section IV, and is shown to outperform a previously benchmarked version of TAH.

II. SYSTEM MODEL

This section formulates the vehicle scheduling problem, and explains the difficulty in obtaining exact optimal solutions to it. The generic class of strategies known as Travel Advance Heuristics (TAH) are described, which have been shown in prior literature to produce feasible solutions with low computational effort. The development of the scheduling methods throughout this paper assumes that resources are allocated to vehicles on an exclusive basis, i.e., each resource can accommodate only one vehicle at a time.

[1]Harshad Khadilkar is with TCS Research (Tata Consultancy Services), Mumbai 400093, India harshad.khadilkar@tcs.com

A. Optimal vehicle scheduling problem

The transportation network is modelled as a graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ denotes the set of all nodes, and $\mathcal{E}$ denotes the set of all edges. A set of vehicles $\mathcal{V}$ is to be scheduled through this network, which implies that vehicles $v_i \in \mathcal{V}$ must be allotted time slots at successive nodes and edges, such that they can move from their respective origins to destinations via predefined routes (sequence of nodes). Each pair of nodes is connected by at most one edge, and thus routes also define the sequence of edges to be traversed. Each node $n_j \in \mathcal{N}$ and edge $e_k \in \mathcal{E}$ is assumed to be composed of one or more parallel (equivalent) resources, denoted by r_m^{nj} and r_p^{ek} respectively, where $m \in \{1, \ldots, R_j^n\}$ and $p \in \{1, \ldots, R_k^e\}$.

Let us define the arrival time of a vehicle v_i at node n_j by $t_i^a(n_j)$, and its departure time to be $t_i^d(n_j)$. Complementarily, the arrival time to and departure time from an edge e_k is denoted by $t_i^a(e_k)$ and $t_i^d(e_k)$ respectively. If e_k is traversed upon leaving n_j, then $t_i^a(e_k) = t_i^d(n_j)$. If the next node after e_k is n'_j, then $t_i^d(e_k) = t_i^a(n'_j)$. For simplicity, it is assumed that all parallel resources at a node are accessible from all resources at adjoining edges. Finally, we define the binary variables $b_m^{nj}(i)$ and $b_p^{ek}(i)$ to be equal to 1 if v_i is allocated to resources r_m^{nj} and r_p^{ek} at respective nodes or edges, and 0 otherwise. Each vehicle v_i has an earliest start time on its journey (arrival time at first node $n_j^{i,\text{ini}}$) given by T_i, and its computed finishing time (departure from last node) is denoted by f_i. Its minimum halt time at node n_j is given by $H_i(n_j)$, and minimum travel time on edge e_k is given by $W_i(e_k)$. Under these assumptions, the generalised scheduling problem is defined as follows:

$$\text{Minimise} \sum_i f_i \tag{1}$$

subject to the following time constraints,

$$t_i^d(n_j) - t_i^a(n_j) \geq H_i(n_j) \tag{2}$$

$$t_i^d(e_k) - t_i^a(e_k) \geq W_i(e_k) \tag{3}$$

$$t_i^a(n_j^{i,\text{ini}}) \geq T_i \tag{4}$$

and the following resource allocation constraints,

$$\sum_m b_m^{nj}(i) = 1 \tag{5}$$

$$\sum_p b_p^{ek}(i) = 1 \tag{6}$$

and if v_{i1}, v_{i2} are both allotted the same resource and v_{i1} is scheduled to occupy it before v_{i2}, then

Doubt

$$t_{i1}^d(n_j) \geq t_{i2}^a(n_j) \text{ if resource at node } n_j \tag{7}$$

$$t_{i1}^d(e_k) \geq t_{i2}^a(e_k) \text{ if resource at edge } e_k \tag{8}$$

Mixed integer linear programming versions that avoid the nonlinear nature of the conditional constraints (7) and (8) have been proposed [17], but have achieved limited success in terms of computational time and maximum problem size.

B. Travel Advanced Heuristic (TAH) class of strategies

Given the computational limitations of exact methods for solving large instances of the vehicle scheduling problem, a popular heuristic solution involves advancing one vehicle by one node in a given *move*. This strategy is known as a travel advance heuristic [15], [16]. The choice of vehicle to be moved in each step varies from one version of TAH to another. The effectiveness of TAH is strongly dependent on the choice of move, as decided by the function call *choose*(v_i) in Step 4 of Algorithm 1. Note that the last step in each iteration (Step 22) sets the arrival time for the vehicle at the next node. This time is set based on the latest information at the time the move is made. However, intervening moves for other vehicles may make the current move infeasible at the desired time. In such an event, the algorithm *backtracks*, or requests a delayed arrival at the current node by delaying departure from the previous node. The magnitude of requested delay is given by $\delta_i(n_j)$. The current location of each vehicle v_i is denoted by $\ell_i \in \mathcal{N}$, which is the node up to which v_i has been scheduled. A generic notation for the previous node (n_j^*), previous edge (e_k^*) and the next node (n'_j) is used in Algorithm 1.

Algorithm 1 Travel Advance Heuristic (Generic TAH)

1: **Input:** T_i, $H_i(n_j)$, $W_i(e_k)$
2: Set $b_m^{nj}(i) = b_p^{ek}(i) = t_i^d(n_j) = \delta_i(n_j) = 0$, $t_i^a(n_j^{i,\text{ini}}) = T_i$, $t_i^a(n_j \neq n_j^{i,\text{ini}}) = 0$ and $\ell_i = n_j^{i,\text{ini}}$
3: **while** scheduling not complete,
4: Pick the next move using *choose*(v_i)
5: Search for a resource at n_j which is available from $t_i^a(n_j)$ to $t_i^a(n_j) + H_i(n_j) + \delta_i(n_j)$
6: **if** not available, [backtrack:]
7: Reset $\delta_i(n_j) = 0$
8: Find earliest time $t_{\min}$ at which a resource at n_j is available for a duration of $H_i(n_j)$
9: If $n_j \neq n_j^{i,\text{ini}}$ set delay at previous node, $\delta_i(n_j^*) = t_{\min} - t_i^a(n_j)$, else set $t_i^a(n_j) = t_{\min}$
10: Erase previous $b_m^{nj*}(i) = b_p^{ek*}(i) = t_i^d(n_j^*) = 0$
11: **continue** to Step 3
12: **end if**
13: Set $t_i^d(n_j) = t_i^a(n_j) + H_i(n_j) + \delta_i(n_j)$
14: Search for a resource at the next edge e_k which is available from $t_i^d(n_j)$ to $t_i^d(n_j) + W_i(e_k)$, and one at n'_j from $t_i^d(n_j) + W_i(e_k)$ to $t_i^d(n_j) + W_i(e_k) + H_i(n'_j)$
15: **if** not available, [extend at current node:]
16: increase $t_i^d(n_j)$ until such resources are available
17: **if** $t_i^d(n_j)$ cannot be extended,
18: Backtrack using Step 6
19: **end if**
20: **end if**
21: Set resource allocations $b_m^{nj}(i)$ and $b_p^{ek}(i)$
22: Set $t_i^a(n'_j) = t_i^d(n_j) + W_i(e_k)$ and update $\ell_i = n'_j$
23: **end while**
24: **Output:** Resource allocations and arrival/departure times for all vehicles

Should a vehicle be *backtracked*, i.e., one or more moves be rolled back as described in Step 6, the algorithm may need to recompute values for a move that was completed several iterations earlier. Other moves that were completed between the last move for v_i and its current move are not changed. These values appear as constraints (7)-(8) when v_i is moved forward again. There are two primary causes for backtracking to fail in the TAH setting, leading to deadlock. The first cause is a race condition between two vehicles, where each vehicle desires to move into the resource occupied by the other, leading to neither finding a feasible move. Such events are frequently triggered in perfectly symmetric problem instances, as demonstrated in Section IV. A second cause, not relevant in the current context, is when TAH is applied to a real-time setting where vehicles cannot be backtracked beyond their physical locations in the network [5].

III. PROPOSED MOVE SELECTION LOGIC

TAH iteratively *moves* vehicles through resource allocations and arrival/departure times. The order in which moves are made is driven by Step 4 in Algorithm 1, and strongly affects the number of backtracks required (see Section IV). The purpose of this section is to define and analyse a form for *choose*(v_i) that prevents backtracking to the extent possible, without requiring any future move exploration.

A. Move selection logic

The procedure *choose*(v_i) is called by Algorithm 1 in each iteration, and moves the chosen vehicle to the next node on its route, if feasible. Implicit in this procedure is the notion of the current location of each vehicle. We have defined the current location ℓ_i of a vehicle v_i to be n_j, if the vehicle has been assigned an arrival time $t_i^a(n_j)$ at n_j but has not been assigned any resources. The *occupancy* of a node n_j is defined to be the number $o(n_j)$ of vehicles whose current location is n_j. The number of resources at a node n_j is equal to R_j^n. Since the physical capacity R_j^n only limits the number of vehicles simultaneously located at n_j while $o(n_j)$ may include future arrivals and departures, $o(n_j) \geq 0$ but is not upper-bounded. The number of unassigned resources at n_j is given by $x(n_j) = R_j^n - o(n_j)$. Similar to $o(n_j)$, these unassigned resources do not relate to physical occupancy, but to the number of vehicles currently scheduled up to n_j. We define the *criticality* of the system by X_n such that,

$$X_n = \min_j x(n_j). \tag{9}$$

Finally, we account for any static priorities p_i for vehicles, if available. In case of a resource conflict, a vehicle with a lower value of p_i is preferentially allotted a resource. The priority of a node is derived from the priority of vehicles currently located at the node,

$$p(n_j) = \min_i \{p_i \,|\, \ell_i = n_j\}$$

With these quantities defined, the move selection logic *choose*(v_i) is given by Algorithm 2. Note that the consideration of priorities p_i is a second-level condition, with the main condition for choosing a move being the criticality X_n.

Algorithm 2 Critical-First logic for *choose*(v_i)

Input: R_j^n, ℓ_i, $t_i^a(n_j)$, $t_i^d(n_j)$ $\forall\{i,j\}$
Compute state of system $x(n_j)$ and X_n
Find critical nodes $\mathcal{N}_c \subset \mathcal{N}$ satisfying $x(n_j) = X_n$
Minimum priority in $\mathcal{N}_c$: $p_c = \min\{p(n_j) | n_j \in \mathcal{N}_c\}$
if not unique ($|\mathcal{N}_c| > 1$): *Break the tie using priorities*
    Subset $\mathcal{N}_p \subset \mathcal{N}_c$ composed of $\{n_j \in \mathcal{N}_c | p(n_j) = p_c\}$
    First departure: $d_{\min} = \min_{v_i:\ell_i \in \mathcal{N}_p}\{t_i^d(n_j)\}$
else $\mathcal{N}_p = \mathcal{N}_c$
end if
if $|\mathcal{N}_p| > 1$: *Break the tie using earliest departure*
    $\mathcal{N}_d \subset \mathcal{N}_p$ with $\{n_j \in \mathcal{N}_p | \min_{v_i:\ell_i=n_j} t_i^d(n_j) = d_{\min}\}$
else $\mathcal{N}_d = \mathcal{N}_p$
end if
if $|\mathcal{N}_d| > 1$: *Pick any one of remaining nodes*
    Set $\mathcal{N}_{\text{chosen}} = \{\text{any } n_j | n_j \in \mathcal{N}_d\}$
else $\mathcal{N}_{\text{chosen}} = \mathcal{N}_d$
end if
Output: Return v_i with lowest p_i located $\ell_i = \mathcal{N}_{\text{chosen}}$

B. Deadlock/backtrack prevention property

TAH encounters deadlock if the next chosen move is infeasible because (i) vehicle v_i finds all resources at the next node occupied by other vehicles, and (ii) these other vehicles can only release their current resources if they move into the resource currently occupied by v_i. Prop. 1 defines a necessary (but not sufficient) condition for encountering a deadlock in the current context.

Prop. 1: If TAH as given in Algorithm 1 encounters deadlock/backtracking, then the proposed move has attempted to change the criticality from $X_n \leq 0$ to $X_n \leq -1$.

The proof follows by noting that vehicle v_i must find no free resources at the next node in order to hit deadlock. It follows that all resources at the next node must be occupied before the move begins ($X_n \leq 0$ at the start of the move). The proposed move attempted to change the critical state to $X_n \leq -1$, since the destination node would contain at least one vehicle more than its resource capacity R_j^n.

As the node occupancy $o(n_j)$ may exceed the number of resources at n_j with non-overlapping arrival/departure times for the vehicles, the criticality may drop below -1 without entering deadlock. Hence the reverse of Prop. 1 is not necessarily true. For the purposes of this study, it is sufficient to show that the proposed version of TAH with the critical-first logic (TAH-CF) never attempts to reduce the criticality below 0. Prop. 2 ensures this condition under some assumptions about the initial state of the system.

Prop. 2: If (i) the criticality before start of scheduling satisfies $X_n \geq 1$, and (ii) TAH-CF as formed by Algorithm 1 and Algorithm 2 is applied, then the scheduling can be completed with no deadlock/backtracking.

If Prop. 2 can be proved for $X_n = 1$, it is trivially proved

for $X_n > 1$. Therefore, we consider the case $X_n = 1$. Based on the initial state of the system, let $\mathcal{N}_1$ be the set of nodes satisfying $x(n_j) = 1$, and $\mathcal{N}_1^C$ be the complement of this set. Since it is assumed that $X_n = 1$, nodes in $\mathcal{N}_1^C$ must satisfy $x(n_j) > 1$. The first move is chosen from one of the nodes in $\mathcal{N}_1$, according to Algorithm 2. The move may have as its destination another node in $\mathcal{N}^C$. Since such a node satisfies initial condition $x(n_j) > 1$, its new state is $x(n_j) \geq 1$. This implies that the new criticality is $X_n = 1$, and we return to a situation analogous to the initial state.

Alternatively, the first move may have as its destination a node in $\mathcal{N}_1$. In this case, the destination node will now have $x(n_j) = 0$ and the criticality will be $X_n = 0$. Since all other nodes in the system have $x(n_j) \geq 1$, this destination node must be chosen for the next move by Algorithm 2. Proceeding logically forward, until moves are made that end in nodes initially satisfying $x(n_j) = 1$, the chosen moves must be unique, and the criticality must remain $X_n = 0$. At some point, a move will end in a node with $x(n_j) > 1$, or the vehicle will complete its journey and leave the system. The criticality will then become some $X_n \geq 1$, and the process will repeat. The system will never reach $X_n = -1$, thus proving the proposition.

It should be noted that not all problem instances will satisfy the initial condition required by Prop. 2, and thus backtrack/deadlock prevention cannot be guaranteed in all instances. In instances where the initial state is $X_n \leq 0$, Algorithm 2 preferentially chooses moves that tend to increase this value. If the system achieves $X_n \geq 1$ at some iteration, then Prop. 2 guarantees that there will be no backtracking/deadlocks from that point forward.

C. Indication of schedule quality

Each iteration of Algorithm 1 effectively solves a simpler version of the optimisation problem (1), where the objective is to minimise $t_i^a(n'_j)$ for the next move as selected by *choose*(v_i). The constraints (2), (3), (5) and (6) are unaffected, while (4) is applicable only if the next move is the first node for a given vehicle. The ordering constraints (7) and (8) are enforced only for values that have already been set. Arrival/departure times and resource allocations for moves that have already been completed are handled as hard constraints for subsequent moves. Therefore, the feasible search space reduces as the number of iterations increases.

Any backtracks in the schedule for one vehicle erase the arrival/departure times and resource allocations for that vehicle at that step only, and move the vehicle to a prior point. When the vehicle is subsequently brought forward again, it must find resources and times in a more constrained state space than its initial forward journey. This suggests that reducing the number of backtracks in the execution of Algorithm 1 will directly reduce the number of constraints that the local optimisation iterations are solved under. Fewer constraints should allow the TAH to produce better local solutions, and this in turn should improve the overall objective value (1). Prop. 2 shows that under the correct initial conditions, Algorithm 2 ensures the minimum number of backtracks (none) among all strategies falling within TAH. Therefore, the schedule quality as produced by critical-first scheduling is likely (not guaranteed) to be better than that produced by other forms of TAH. Section IV tests this hypothesis for three cases increasing in size.

D. Demonstration using hypothetical instance

In the railway context, the nodes of the network are represented by stations, and the edges by inter-station track sections. Each station has resources in the form of one or more tracks with associated platforms for loading and unloading passengers and freight. Each inter-station track section may contain one or more parallel tracks for allowing train movement from one station to the next. The network model is explained in greater detail in a prior study [18].

A hypothetical railway network is shown in Fig. 1. The structure of the network is linear, with four stations (nodes) connected in series by three track sections (edges). Stations Alpha and Delta contain four resources each, while Bravo and Charlie have two each. The track sections have only one resource each. Six trains (vehicles) are to be scheduled, with three trains initially located at each of the two terminal nodes. Each train is available for starting immediately, and needs to move to the other end of the network before exiting the system. All six trains are assumed to have equal priority. Algorithm 1 and Algorithm 2 are to be used to produce schedules of movement for all six trains.

Fig. 1 shows the initial state $x(n_j)$ of each node as numbers above the names. Alpha and Delta have $x(n_j) = 1$, while Bravo and Charlie have $x(n_j) = 2$. The criticality is $X_n = 1$, satisfying the initial condition as given in Prop. 2. The first node set $\mathcal{N}_c$ from Algorithm 2 is composed of Alpha and Delta. Since all trains are of equal priority and are available at the same time, any one train may be picked randomly for the first move. The next four moves are demonstrated in Fig. 2, and the algorithm proceeds to completion. The results are discussed in the next section.

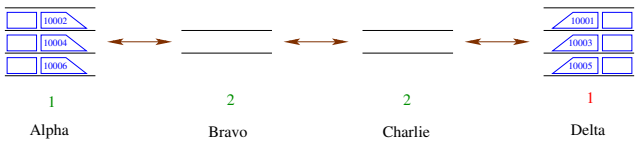

Fig. 1. A hypothetical railway network consisting of four stations (nodes) and three inter-station sections (edges). Three trains are present at each of the terminal nodes, and must travel to the opposite end.

IV. RESULTS

This section demonstrates the application of TAH-CF to three instances of railway networks. As explained in Section III-D, nodes of the network are represented by railway stations, and edges by track sections between stations. Each station and track section has one or more parallel resources (railway tracks). The results of TAH-CF are compared with a more basic version of TAH, adapted from prior literature [5]. This heuristic, labelled *fixed priority* (TAH-FP), enforces a strict priority order (if priorities p_i are defined) for choosing

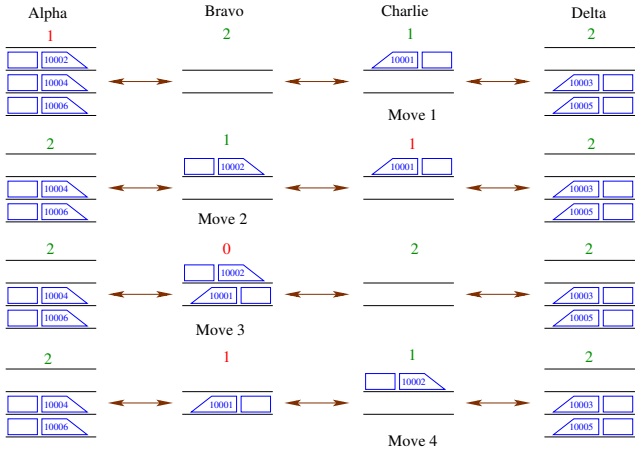

Fig. 2. Illustration of the first four moves performed by TAH-CF in the hypothetical example from Fig. 1.

vehicle (train) moves. It differs from TAH-CF solely through omitting Step 3 of Algorithm 2, instead setting $\mathcal{N}_c = \mathcal{N}$. The priority-based selection in Step 5 is the first discriminatory portion of TAH-FP. The result of *choose*(v_i) in TAH-FP is the earliest departing ($d_{\min}$) train of the lowest numerical priority (p_c) among all nodes, as opposed to just the critical node. TAH-FP involves searching through the entire set of trains to find the next move. By contrast, TAH-CF searching through the nodes to find the critical node, followed by a search through a smaller set of trains at the critical node. Therefore, the number of computational steps in the two methods is comparable.

A. Test case 1: small hypothetical network

A scheduling problem based on the network shown in Fig. 1 is defined. The minimum halt time $H_i(n_j) = 1$ hour $\forall i, j$, and the minimum travel time $W_i(e_k) = 1$ hour $\forall i, k$. The schedule computed by TAH-CF is shown in Fig. 3 in the form of a time-space chart, where the x-axis represents time and the y-axis represents spatial location along the linear network. The resources available at each station are marked as dotted horizontal lines at each station location. The solid lines represent train movements through the network, with horizontal portions representing halts at stations, and oblique portions representing travel to the next station. The left (right) hand side of each horizontal segment represents the arrival (departure) time at that station. For example, the first running train arrives at Alpha at $t = 0$, departs from Alpha at $t = 1$, and subsequently arrives at Bravo at $t = 2$. Since each edge contains a single resource, no two oblique lines may cross each other in between stations.

The same scheduling instance is then attempted by TAH-FP, which runs into deadlock and is unable to find a feasible schedule. This happens because of the symmetry in the network topology and the train schedules, combined with the bottleneck created by the two resource-constrained stations Bravo and Charlie. Moves made by trains leaving from Alpha are exactly mirrored by those made by trains leaving from Delta, and the two sets are unable to reach consensus for traversing the track section between Bravo and Charlie.

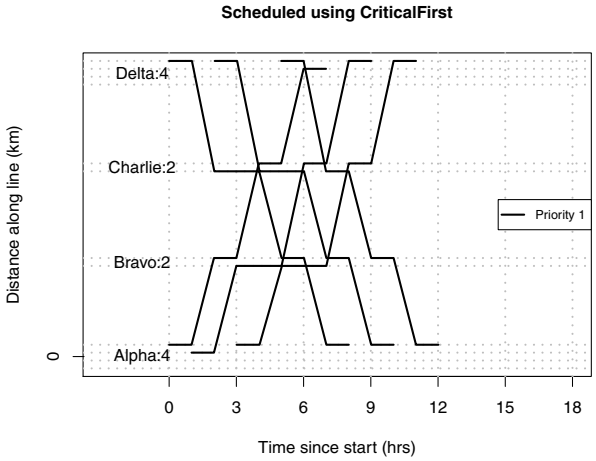

Fig. 3. Schedule produced by TAH-CF for a hypothetical railway network with 6 trains (vehicles) and 4 stations (nodes).

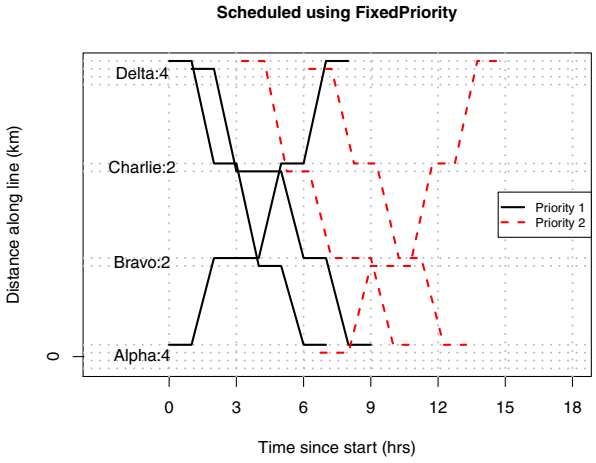

Fig. 4. Schedule produced by TAH-FP for a hypothetical railway network, after asymmetry is artificially introduced by changing train priorities.

When asymmetry is artificially introduced in the problem by defining three of the six trains to have priority $p_i = 1$ and three trains to have priority $p_i = 2$, TAH-FP is able to find a feasible schedule. The result is shown in Fig. 4. Note that the three trains with $p_i = 1$ have shorter journey times than those with $p_i = 2$. A comparison with Fig. 3 shows that while the last train in TAH-CF finished its journey at 12 hours, the last train in TAH-FP finished in 15 hours.

B. Test cases 2 and 3: portions of Indian Railway network

For a more realistic comparison, both algorithms are tested on data obtained for two portions of the Indian Railway network. The first data set pertains to the Konkan railway line which runs over 800 km on the Indian west coast. This network is a single linear arc with 59 stations (nodes). The second data set pertains to a network of railway lines between Mumbai and New Delhi (MUM-DEL). A schematic of the topology is shown in Fig. 5. The network contains 5 linear arcs joined at 6 junction stations as shown, while Mumbai and New Delhi are the terminal stations. The train routes

and halt times are obtained from public data sources [19]. The travel times on edges are derived using methodology published in literature [18]. In order to control for problem size, Table I compares a modified version of the objective (1). The average delays are computed by subtracting the finishing time of each train according to the timetable, and dividing by the number of trains to be scheduled.

The Konkan data set contains 85 trains spanning 6 priority levels, with a total of 2709 station arrival/departure events. Table I shows that the average delay in the schedule produced by TAH-FP is 4 minutes, while it is 3 minutes for TAH-CF. The number of backtracks required by TAH-CF is just over half of those required by TAH-FP, but is not equal to 0. This is because the initial state of the system does not satisfy the constraint $X_n \geq 1$ as defined in Prop. 2. Table I also shows that the maximum delay assigned to any train by TAH-CF is lower than that by TAH-FP. There is a tradeoff between the delays assigned to trains of different priority. TAH-CF assigns a higher value of delay to trains with $p_i = 1$, while reducing the delays faced by trains with $p_i = 6$.

A similar delay comparison is seen in the MUM-DEL instance. However, both TAH-FP and TAH-CF require the same number of backtracks. This reflects the differences in edge capacities in the Konkan and MUM-DEL cases. While all edges in the former contain a single railway track, a majority of edges in the latter contain 2 railway tracks. The higher capacity leads to fewer backtracks (0.8% of instances) for MUM-DEL. Incidentally, TAH-CF requires 6,100 moves to reach $X_n \geq 1$ for the first time, after which there is no backtracking. TAH-FP requires backtracks throughout the procedure, and backtracks in the latter stages add more delays than those in the earlier stages of scheduling.

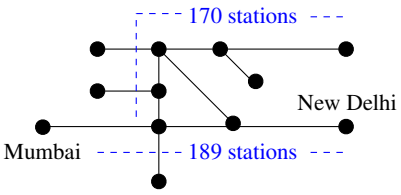

Fig. 5. Topology of the Mumbai – New Delhi (MUM-DEL) railway network. The two longest arcs contain 357 of the 418 nodes (stations).

V. CONCLUSIONS

This paper proposed a modification to the basic version of travel advance heuristics for transportation network scheduling. Move selection using critical-first was shown to (i) reduce the number of deadlocks/backtracks, and (ii) improve schedule quality measured in terms of travel times on preplanned routes. A theoretical guarantee for deadlock/backtrack prevention was proven, assuming certain conditions on the initial state of the system. The proposed logic does not require any future move exploration to provide this guarantee. Analytical indications of the improved schedule quality in generic scheduling instances were also outlined. Extensions of this work include its application to domain areas other than railways, as well as to real-time scheduling.

TABLE I

PERFORMANCE COMPARISON BETWEEN TAH-FP AND TAH-CF.

Network	Metric	TAH-FP	TAH-CF
Konkan rail	Network	Linear, 59 nodes, 740 km	
	Vehicles (trains)	$\|\mathcal{V}\| = 85$, 6 priorities	
	Total events	2709	
	Backtracks	2.5%	1.4%
	Average delay $\sum(f_i - f_{i,\text{sch}})/\|\mathcal{V}\|$	4 min	3 min
	Maximum delay $\max(f_i - f_{i,\text{sch}})$	20 min	13 min
	Avg. for $p_i = 1$	1 min	2 min
	Avg. for $p_i = 6$	6 min	4 min
MUM-DEL	Network	Fig. 5, 418 nodes, 2753 km	
	Vehicles (trains)	$\|\mathcal{V}\| = 1027$, 5 priorities	
	Total events	40952	
	Backtracks	0.8%	0.8%
	Average delay $\sum(f_i - f_{i,\text{sch}})/\|\mathcal{V}\|$	4 min	1 min
	Maximum delay $\max(f_i - f_{i,\text{sch}})$	256 min	147 min
	Avg. for $p_i = 1$	0 min	2 min
	Avg. for $p_i = 5$	3 min	1 min

REFERENCES

[1] S Liu and E Kozan, "Scheduling trains as a blocking parallel-machine job shop scheduling problem", Computers & Operations Research, vol. 36 (10), 2009, pp. 2840–2852.

[2] A Mascis and D Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints", European Journal of Operational Research, vol. 143 (3), 2002, pp. 498–517.

[3] A D'Ariano, D Pacciarelli and M Pranzo, "A branch and bound algorithm for scheduling trains in a railway network", European Journal of Operational Research, vol. 183, 2007, pp. 643–657.

[4] F Corman and L Meng,"A review of online dynamic models and algorithms for railway traffic management", Journal of Intelligent Transportation Systems, vol. 16 (3), 2015, pp. 1274–1284.

[5] S Sinha, S Salsingikar and S SenGupta, "Train scheduling using an iterative bi-level hierarchical approach", International Conference on Railway Operations Modelling and Analysis, Mar 2015.

[6] X Cai and C Goh, "A fast heuristic for the train scheduling problem", Computers & Operations Research, vol. 21, 1994, pp. 499–510.

[7] J Adams, E Balas and D Zawack, "The shifting bottleneck procedure for job shop scheduling", Mgmt. Sci., vol. 34 (3), 1988, pp. 391–402.

[8] S Mackenzie, "Train scheduling on long haul railway corridors", PhD thesis, University of South Australia, 2010.

[9] E Coffman, M Elphick and A Shoshani, "System deadlocks", Computing Surveys, vol. 3 (2), 1971, pp. 67–78.

[10] M Dorfman and J Medanic, "Scheduling trains on a railway network using a discrete event model of railway traffic", Transportation Research Part B, vol. 38, 2004, pp. 81–98.

[11] J Pachl, "Avoiding deadlocks in synchronous railway simulations", Int. Seminar on Railway Operations Modelling and Analysis, Mar 2007.

[12] P McFarlane and H Balakrishnan, "Optimal control of airport push-backs in the presence of uncertainties", American Control Conference, May 2016.

[13] H Khadilkar and H Balakrishnan, "A network congestion control approach to airport departure management", American Control Conference, May 2012.

[14] I Simaiakis and H Balakrishnan, "Design and simulation of airport congestion control algorithms", American Control Conference, 2014.

[15] X Cai, C Goh and A Mees, "Greedy heuristics for rapid scheduling of trains on a single track", IIE Transactions, vol. 30, 1998, pp. 481–493.

[16] J Medanic and M Dorfman, "Efficient Scheduling of Traffic on a Railway Line", Journal of Optimization Theory and Applications, vol. 115 (3), 2002, pp. 587–602.

[17] G Sahin, R Ahuja and C Cunha, "Integer programming based solution approaches for the train dispatching problem", Tech. Rep., Sabanci University, 2010.

[18] H Khadilkar, "Data-Enabled Stochastic Modelling for Evaluating Schedule Robustness of Railway Networks", Transportation Science, *articles in advance*, Sep 2016.

[19] India Rail Info, "Train timetable and operational data archives", http://indiarailinfo.com, Retrieved January 2015.