


Using A* for Optimal Train Routing on Moving Block Systems

Stefan Engels¹ ✉ 

Chair for Design Automation, Technical University of Munich, Germany

Robert Wille ✉ 

Chair for Design Automation, Technical University of Munich, Germany

Abstract

Modern control systems based on Moving Block allow for shorter headways and higher capacity on existing railway infrastructure. At the same time, few algorithms for optimal routing on networks equipped with such modern control systems exist. Previous methods rely on Mixed Integer Linear Programming (MILP) and face a trade-off between model size and accuracy, especially considering comparably complex and nonlinear headway constraints as well as train dynamics. With this work, we propose a complementary approach based on A*. Under a reasonable and easy assumption on train driver behavior, we propose a solution encoding and state space that is flexible concerning the choice of search algorithm and the modeling detail. The applicability is showcased on a small benchmark set. The implementation is available open-source as part of the *Munich Train Control Toolkit* (MTCT) on GitHub at <https://github.com/cda-tum/mtct>.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases ETCS, Train Routing, Moving Block, A*, Munich Train Control Toolkit

Supplementary Material *Software (Source Code)*: <https://github.com/cda-tum/mtct>

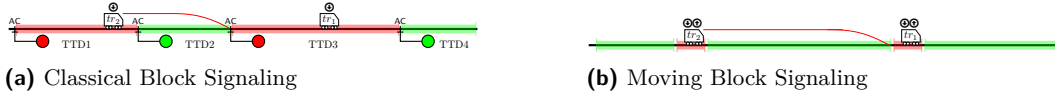
1 Introduction

Railway traffic plays a vital role in the future of sustainable transportation. Due to long braking distances, trains cannot operate on sight (if they should travel at a reasonable speed). Instead, signaling systems are needed for safe separation. In the past, these systems have been specified on a national level. With increasing international rail traffic, control systems have been unified across Europe, leading to the *European Train Control System* (ETCS). Other common systems are the *Chinese Train Control System* (CTCT), and *Positive Train Control* (PTC) [11] as well as *Communication Based Train Control* (CBTC) for metro systems [15].

Today, many railway lines operate at their capacity limit. Building new infrastructure is costly and time-consuming. Complementary to that, the specifications of control systems have been improved. Modern versions (e.g., based on *Hybrid Train Detection* or *Moving Block*) allow for shorter headways. However, the capacity can only be used if they are adequately considered in the infrastructure planning process. To cope with the additional degree of freedom, new design tasks arise [4].

Secondly, modern headway concepts need to be considered when creating a timetable. Most research on timetable optimization and train routing focuses on networks equipped with classical control systems [1]. Only limited research considers modern Moving Block systems [14, 10, 6]. Engels and Wille [7] show that these solutions can also be used as part of an optimization pipeline for design tasks [4] within the context of Hybrid Train Detection systems.

¹ Corresponding author



■ **Figure 1** Schematic drawings of different signaling principles [6]

Previous methods are mainly based on *Mixed Integer Linear Programming* (MILP). There is always a trade-off between efficiency and accuracy when creating a model. Usually, sensible headway constraints are the most complex to model. For this, discretization and linearization techniques are used.

In this work, we propose an alternative, complementary approach to train routing on Moving Block systems that can operate at an arbitrary level of detail. Any (black-box) simulator can be used to evaluate train movements. Our proposed approach is based on A*. It has already been successfully applied to different areas within the design automation community, e.g., in Nanotechnologies [9]. Even within the rail domain, first solutions based on A* exist [12]. However, their approach cannot easily be generalized to consider train movements and headways at sensible levels of detail due to the choice of solution encoding. With this work, we propose a different approach that incorporates the complex constraints already in the encoding. Because of this, more complex relations can be incorporated without further complicating the A* search.

The remainder of this work is structured as follows. Section 2 reviews the relevant concepts of train control systems and A* search. Under an additional but sensible assumption, a solution encoding is proposed in Section 3. Sections 4 and 5 showcase how this can be used to design an optimization algorithm for train routing. Section 6 provides a proof of concept. Finally, Section 7 concludes this work.

2 Background

This section reviews the relevant concepts of train control systems and informed search algorithms such as A*.

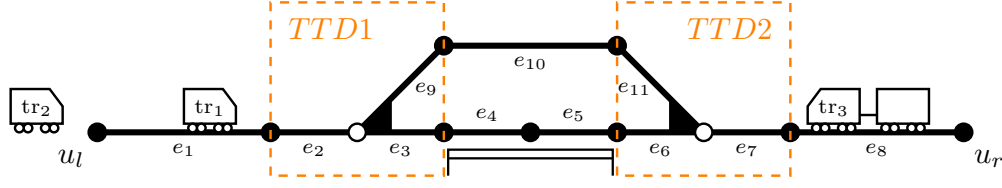
2.1 Train Control Principles

Classically, railway networks are separated into fixed blocks. To detect the status of a given block, tracks are equipped with *Trackside Train Detection* (TTD) hardware, e.g., axle counters. A train may only proceed into the next section if the entire block is unoccupied.

► **Example 1** ([6]). Consider two trains following each other on a single track as depicted in Figure 1a. Train tr_2 can only move until the end of $TTD2$. It cannot enter $TTD3$ because it is still occupied and, hence, might have to slow down in order to be able to come to a full stop before entering the occupied block section.

Trains equipped with a *Train Integrity Monitoring* (TIM) system can safely report their position. In that case, trains can follow each other at absolute braking distance without the need for any block sections². *Moving Block* control systems make use of this concept and are already implemented on some metro networks.

² At the same time, we allow the existence of some TTD sections. They can be used to, e.g., provide basic flank protection around switches.



■ **Figure 2** Example network

► **Example 2** ([6]). In contrast to Example 1, consider a moving block control implemented in Figure 1b. Because trains operate at the ideal absolute braking distance, tr_2 can move up to the actual end of tr_1 (minus a little buffer). In particular, it can already enter what has been $TTD3$ previously. Hence, trains can follow each other more closely.

2.2 Train Routing

This work considers time-optimal train routing on railway networks equipped with a Moving Block control system. For this, we are given general timetable requests. Routing is the task of deciding on which specific track to use and when trains are moving; all constraints of the signaling system need to be fulfilled. In particular, we consider the tasks on a microscopic level. In theory, many different objectives are possible. We aim to optimize the respective travel time.

► **Problem 3** (Optimal Train Routing (on Moving Block Systems)). Given:

- A railway network with vertices V and (directed) edges E as described in Appendix A.
- A set of trains, where w_i denotes the weight of importance of train tr_i
- Demands for every train tr consisting of:
 - an entry node $u_{in}^{(tr)} \in V$ together with a desired entry time interval $[\underline{t}_{in}^{(tr)}, \bar{t}_{in}^{(tr)}]$ and initial velocity $v_0^{(tr)}$,
 - an exit node $u_{out}^{(tr)} \in V$ together with a desired exit time interval $[\underline{t}_{out}^{(tr)}, \bar{t}_{out}^{(tr)}]$ and exiting velocity $v_{out}^{(tr)}$, as well as,
 - a list of stations $\mathcal{S}_1, \dots, \mathcal{S}_{m_i} \subset E$ together with stopping requests, i.e., a latest arrival time $\bar{\alpha}_i^{(tr)}$, an earliest departure time $\underline{\delta}_i^{(tr)}$ and a minimal stopping time $\Delta t_i^{(tr)}$.

Task: Find a routing satisfying every train's demand, obeying the constraints imposed by a moving block control system, and minimizing the (weighted) exit times.

► **Example 4.** Consider the railway network in Figure 2. It consists of one train station (with two edges and one platform³), as well as two TTD sections to prevent collisions on the respective railway switches. Two trains (tr_1 and tr_2) enter at u_l and traverse toward u_r , whereas tr_3 travels in opposite direction. Assume that tr_1 has to stop at the train station, i.e., stop on edge e_4 or e_5 . Hence, it will travel on the bottom line. Depending on the timings, tr_2 might follow the same path, allowing tr_3 to pass in the opposite direction on the upper track. If tr_3 travels at an earlier or later time, the upper track could also be used for tr_2 to overtake tr_1 in an alternative solution.

³ Of course, the station could also consist of multiple parallel platforms

2.3 A*-Algorithm

A* is an informed search algorithm (classically designed to find shortest paths on directed graphs). It functions similarly to Dijkstra's algorithm but expands toward the destination more quickly by making use of a heuristic approximation of the remaining distance.

Let $G = (V, E, c)$ be a weighted graph, $s_0 \in V$ an initial vertex (also called state in this context), as well as a set of terminal states $\mathcal{T} \subset V$. The task is finding the shortest path from s_0 to any vertex in \mathcal{T} . A* explores the graph starting from s_0 . For every vertex $s \in V$, it keeps track of the shortest path from s_0 to s found so far. We denote the length of this path by $g(s)$. Dijkstra would expand on the vertex with the smallest $g(\cdot)$. However, A* uses a different evaluation function. For every $s \in V$, we apply an efficiently computable heuristic function $h(s)$ approximating the shortest (remaining) distance from s to \mathcal{T} . A* then explores the neighbors $\Gamma(s)$ of the vertex with the smallest combined value $f(s) := g(s) + h(s)$. By doing this, the exploration is guided toward goal states.

■ **Algorithm 1** A*-Algorithm [8]

```

C ← ∅; f(s) ← ∞ for all s ∈ V
add s0 to priority queue (pq); f(s0) ← 0
while pq is not empty:
    select s ∈ arg minσ ∈ pq f(σ)
    C ← C ∪ {s}
    if s ∈ T: return s
    explore all successors Γ(s)
    for s+ ∈ Γ(s) − C:
        add or update s+ to pq
        f(s+) ← min{f(s+), g(s) + c(s, s+) + h(s+)}
return infeasible

```

In theory, we can use any heuristic function h . However, A* is only guaranteed to return optimal values if the heuristic meets certain conditions, namely, that h never overestimates the actual cost. If h satisfies a triangle-like inequality, A* can safely be terminated as soon as the first terminal vertex is explored (as specified in Algorithm 1).

► **Definition 5** (Consistent Heuristic [8]). *A heuristic h is consistent if $h(s) \leq c(s, s^+) + h(s^+)$ for every $(s, s^+) \in E$, and $h(t) = 0$ for every $t \in \mathcal{T}$.*

► **Theorem 6** ([8]). *If h is consistent, then Algorithm 1 returns a minimal path.*

Hart et al. [8] show Theorem 6 for the case of δ -graphs, i.e., if there exists a $\delta > 0$ such that $c(e) \geq \delta$ for every $e \in E$. However, this condition is only used to show termination in a weaker case, when h is just admissible but not consistent. It is easy to verify that Algorithm 1 is optimal even for arbitrary (including negative⁴) edge weights, see, e.g., [2, Theorem 2.9].

3 Solution Encoding

When solving any problem, one must decide on the underlying modeling and its consequences on possible algorithms. A classical approach is to model optimization problems as linear or nonlinear constraints and use general and well-developed solvers for the respective problem

⁴ Note that G cannot have negative cycles if there exists a consistent heuristic.

class. Often, it is possible to formulate *Mixed Integer Linear Programs* (MILP). E.g., Problem 3 in the presented or similar forms were considered in [14, 10, 6]. Finding detailed and efficient models is difficult. With respect to train routing tasks, it is usually most challenging to model adequate headway times induced by the underlying signaling system. To linearize these constraints, previous MILP approaches consider a velocity-extended graph and model simplistic headways only at the vertices of the railway network and do not enforce headways in between. Hence, one must always trade-off between performance and accuracy when choosing the discretization levels.

Previously, A^* was proposed as a different approach to solve design tasks within the infrastructure planning process [12]. At the same time, their approach is also closely related to train routing as described in Problem 3. Peham et al. use a state space discretized by time, hence encountering similar problems. Train dynamics and headways were not properly considered. It is unclear how they could be easily incorporated into their approach since these rather complex constraints would have to be considered while determining possible successor states.

Hence, we propose a fundamentally different encoding, which flexibly incorporates most constraints already at this stage. By doing so, it is also more natural to apply Algorithm 1 on the so-defined states. For this, we reconsider what the relevant decisions to be made by the solution algorithm are. Previous solutions leave much freedom for trains to accelerate and decelerate anytime and anywhere on the network. To reduce the search space, we make an assumption on the behavior of trains.

► **Assumption 7** (Greedy Train Driver Assumption). *Trains always move as fast as the control system allows; they do not slow down unless forced to do so.*

Of course, this assumption changes the feasible region of Problem 3, at the same time, arguably almost only cutting off suboptimal solutions. However, there is no theoretical guarantee because one could design situations in which it is beneficial (also with respect to time) to slow down, e.g., in order not to rear-end slow trains before they might exit the main track. At the same time, under Assumption 7, deciding on the train positions at every time step is no longer necessary. Instead, the train behavior is uniquely determined by the following:

1. the edges (track sections) used by each train,
2. the order of trains on each border vertex and TTD section⁵, and
3. the stop positions of every train within the respective stations.

If this information is decided upon, the train movements can be simulated to determine the objective (or decide that no valid train movements are possible using the predetermined information). Any such fully determined state (in which the simulation succeeds) is a feasible solution to Problem 3, hence, a terminal state (see also Section 2.3).

⁵ the order on single edges can be induced from that

► **Example 8.** Again, consider Example 4 and the corresponding network depicted in Figure 2. We might specify:

- tr_1 and tr_2 use $(e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8)$; tr_3 uses $(e_8, e_7, e_{11}, e_{10}, e_9, e_2, e_1)$.
- Order on u_l and $TTD1$: $(\text{tr}_1, \text{tr}_2, \text{tr}_3)$; order on u_r and $TTD2$: $(\text{tr}_3, \text{tr}_1, \text{tr}_2)$.
- tr_1 stops in the station at the end of e_5 .

Simulating under Assumption 7 might lead to the following movement of trains: tr_1 and tr_3 enter the network at their respective vertices. tr_1 moves all the way to e_5 as fast as possible, stops for the time specified in the timetable request, and continues to its exit vertex u_r . tr_2 enters the network some time after tr_1 and follows tr_2 as close as possible at absolute braking distance. tr_3 moves all the way to e_{10} and stops (even though it does not have a scheduled stop). Only after tr_2 has cleared $TTD1$ will it continue to its exit vertex u_l .

4 Optimization using A*

Having established the relevant decisions in Section 3, we can construct a specific A* approach for Problem 3 under Assumption 7. For this, we need to establish a state space, i.e., a graph on which Algorithm 1 is applied to and sensible heuristic estimates $h(\cdot)$.

4.1 State Space

To use search algorithms, it does not suffice to encode only terminal states. Instead, we must also encode partial solutions that might lead to feasible solutions encoded by terminal states.

► **Definition 9** (Partial Solution). *A partial solution is given by information on every train tr_i consisting of*

- *a list of adjacent edges $(e_1^{(\text{tr}_i)}, \dots, e_{k_i}^{(\text{tr}_i)})$, possibly empty⁶, with $u_{in}^{(\text{tr}_i)}$ if $k_i \geq 1$ and*
- *a list of specific stop positions in stations $\mathcal{S}_1, \dots, \mathcal{S}_{\hat{m}_i}$ for the first $0 \leq \hat{m}_i \leq m_i$ stations⁷. Moreover, the train orders are specified on every TTD section with more than one train traveling on, as well as on the entry and exit vertices.*

In contrast to terminal states (i.e., fully specified movements), $u_{out}^{(\text{tr}_i)} \notin e_{k_i}^{(\text{tr}_i)}$ and $\hat{m}_i \neq m_i$ are possible. When simulating, it is assumed that, in such a case, trains stop at the end of their last specified edge and end their journey on the network. In the objective, the exit time of tr_i is replaced by the time it reaches the end of $e_{k_i}^{(\text{tr}_i)}$ with velocity 0. The *state space*, i.e., the “vertices” of the graph used in Section 2.3, is then given by the set of all (partial) solutions.

► **Example 10.** Again, consider Example 4 and the corresponding network depicted in Figure 2. In contrast to Example 8 the train routes are only specified partially, e.g.,:

- tr_1 uses $(e_1, e_2, e_3, e_4, e_5)$; tr_2 uses no edges; tr_3 uses (e_8, e_7, e_{11}) .
- Order on v_l and $TTD1$: (tr_1) ; order on v_r and $TTD2$: (tr_3) .

In this scenario, tr_1 enters at v_l , moves to the end of e_5 as fast as possible, and stops there permanently; similarly, tr_3 moves to e_{11} . tr_2 does not enter the network at all.

► **Assumption 11.** *We are given a (possibly black-box) simulator that can (under Assumption 7) determine the unique train movements given a partial solution (Definition 9) or return that no feasible trajectory exists (e.g., due to a deadlock situation).*

⁶ $k_i = 0$ is possible

⁷ hence, no stops in stations $\mathcal{S}_{\hat{m}_i+1}, \dots, \mathcal{S}_{m_i}$ are specified in this case

4.2 Transitions

It remains to define transitions between states, i.e., the “edges” of the graph used in Section 2.3. Naturally, the initial state is given by the empty state, i.e., no train enters the network, with objective value 0. The target states \mathcal{T} are all fully specified states corresponding to feasible train movements that respect all timetable requests.

The main idea is to traverse the state space edge by edge. Given any (partial) state s , we obtain possible successor states $s^+ \in \Gamma(s)$ by one of the following:

1. Any single train stops at its current route end (if the respective edge is part of the next scheduled station).
2. Any single train moves to any succeeding edge on the network.
3. Any single train enters the network from outside.

The respective TTD- and vertex-orders are induced by the order in which these transitions were chosen to reach s^+ from the initial state. If a TTD is entered, the train is appended to the respective TTD-order; analog for entry and exit vertices.

► **Example 12.** Consider the state depicted on the left of Figure 3a with three trains. tr_1 and tr_2 have already entered the network, whereas tr_3 ’s route is still empty. There are five possible successor states:

- tr_1 (orange) can either use the current route end to stop in the train station or move onward to the next edge. In this case, tr_1 first enters *TTD2*, hence, the train order on *TTD2* (which was empty before) is now given by (tr_1) .
- tr_2 (blue) can move one edge. Due to the switch, it can either stay on the main track or divert to the left. Since it was already in *TTD1*, no adjustments to the order are needed, and the order on *TTD1* is still given by (tr_1, tr_2) .
- tr_3 (purple) can enter the network on the right, hence, the order on u_r is now given by (tr_3) .

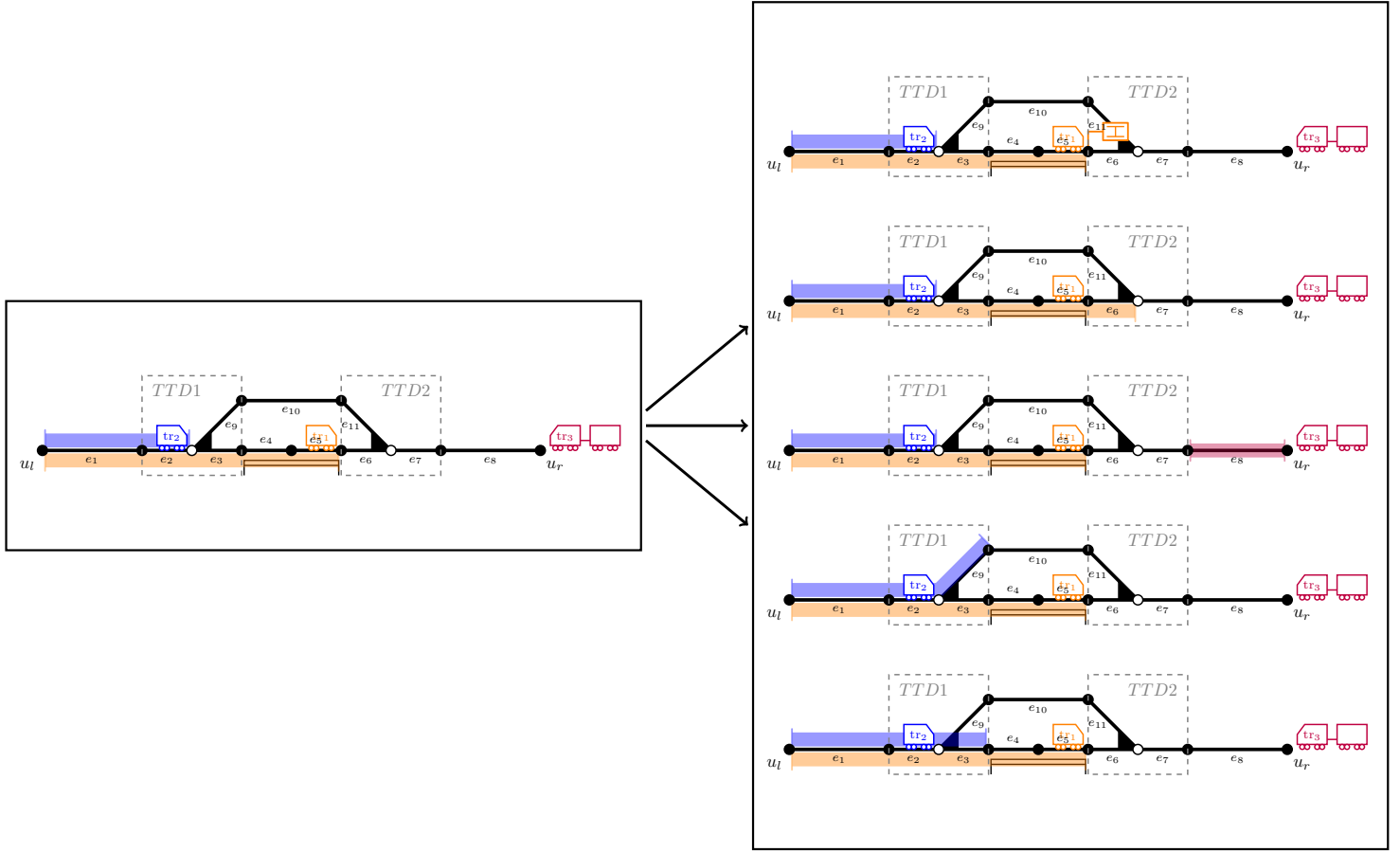
Following this strategy, many intermediate states are created, even if there is only one plausible decision to continue with. Note that trains cannot overtake or cross on single-track lines without turnouts. Hence, train movements of more than one edge can be safely assumed until the next switch is reached. Doing so can reduce the number of states explored by skipping unnecessary intermediate steps.

► **Example 13.** Consider the state depicted on the left of Figure 3b with a single train that has already traversed the first edge. If the train continues, it enters *TTD1*. Since no other train can enter *TTD1* before tr_1 has left, we can move it all the way to the end of the TTD section without skipping any relevant state. The train can either move left or right. Given that decision, there is no routing choice until the next switch, so we can move the train all the way to just before *TTD2*. We cannot safely extend tr_1 ’s route into *TTD2* because another train might still enter *TTD2* before without producing a deadlock. Finally, there are two more possible successors because tr_1 might stop at any of the two possible stop positions of the station.

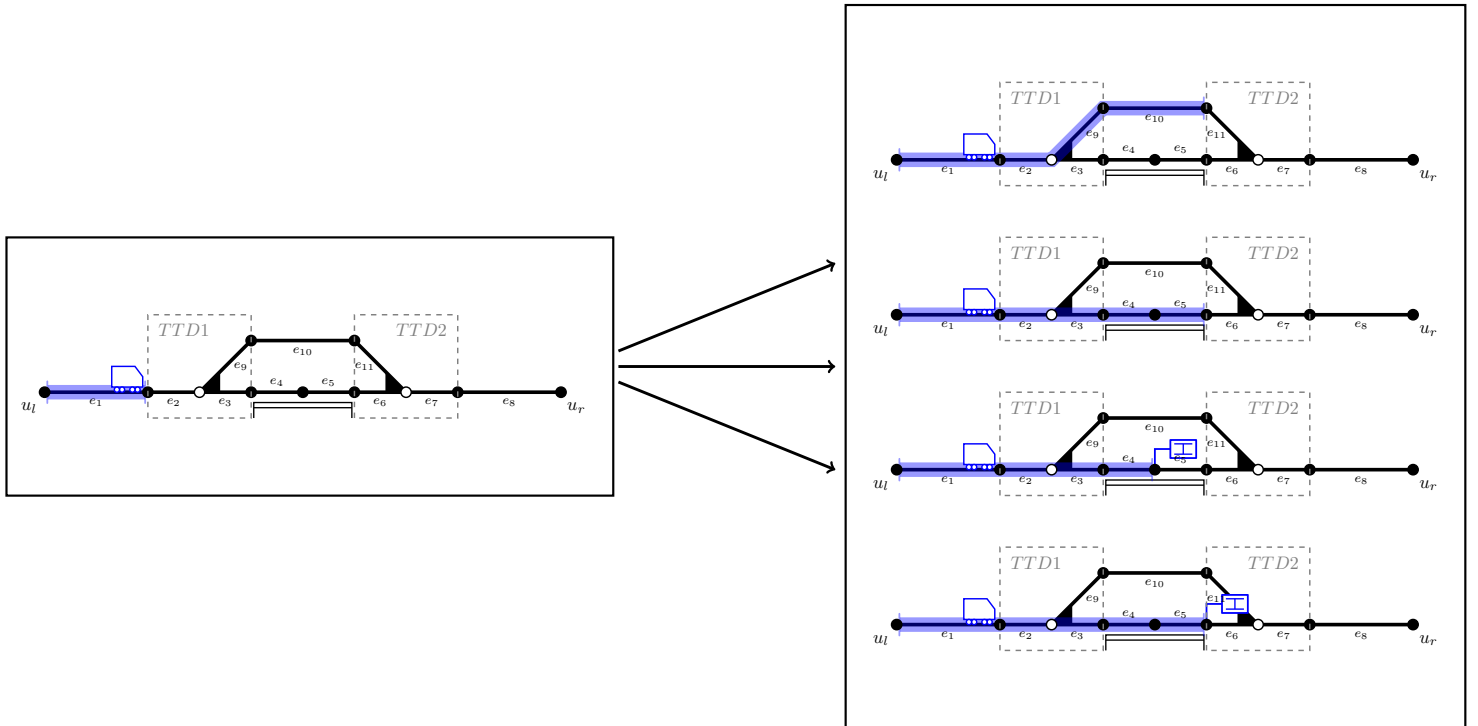
4.3 Suitable Heuristic

For any state s , we can simulate the movements of each train (Assumption 11). Let \bar{t}_i be the time at which tr_i either exits the network or reaches the end of its route as specified by the (partial) state s . We define

$$g(s) := \sum_{i=1}^{\#tr} w_i \cdot \bar{t}_i. \quad (1)$$



(a) Single-Edge Successor State Exploration



(b) Multi-Edge Successor State Exploration

■ **Figure 3** Determination of successor states

Note that for any terminal state $t \in \mathcal{T}$, $g(t)$ coincides with the objective value of the corresponding feasible (but not necessarily optimal) solution to Problem 3. Furthermore, its value does not depend on specific state transitions but is the same regardless of which path was taken to reach a particular state, as all relevant information for simulation is contained in the state itself. We do not define the transition costs $c(s, s^+)$ explicitly, but they are implicitly given by

$$c(s, s^+) = g(s^+) - g(s). \quad (2)$$

Usually, $c(s, s^+) > 0$ will hold. However, for terminal $t \in \mathcal{T}$, $c(s, t) \leq 0$ is theoretically possible because the train is not forced to stop at the exit vertex in contrast to the end of partial routes in intermediate states. However, this is not a problem because we will handle this by choosing a sensible heuristic $h(s)$.

To guide the search algorithm, we need suitable edge weights and an optimistic heuristic estimating the lowest cost $g^*(s, \mathcal{T})$ from any state s to the nearest terminal. For this, we estimate the remaining time for every individual train tr_i with $h_i(s)$. If a train has already reached its exit vertex, we have $h_i(s) = 0$. Otherwise, let $\tau_i^*(p_1, p_2)$ be the minimal running time for tr_i between any two points⁸ on the railway network \mathcal{N} , ignoring headway constraints induced by other trains. Let $\hat{\tau}_i(p_1, p_2)$ be an optimistic approximation, i.e., $0 \leq \hat{\tau}_i(p_1, p_2) \leq \tau_i^*(p_1, p_2)$, which still fulfills the triangle inequality

$$\hat{\tau}_i(p_1, p_2) \leq \hat{\tau}_i(p_1, p_k) + \hat{\tau}_i(p_k, p_2) \quad \forall p_k. \quad (3)$$

For $e = (u_0, u_1) \in E$ a natural and easy to compute choice is

$$\hat{\tau}_i(e) = \hat{\tau}_i(u_0, u_1) = \frac{\text{len}(e)}{\min \left\{ v_{\text{tr}_i}^{(\max)}, v_e^{(\max)} \right\}}, \quad (4)$$

i.e., assuming a train always moves at the maximum speed, disregarding constraints on acceleration and deceleration. In general, we can induce any $\hat{\tau}_i(p_1, p_2)$ as the quickest path from p_1 to p_2 by using the edge timings $\hat{\tau}_i(e)$ defined in Equation (4).

► **Example 14.** Consider the network with edge length and maximal velocities depicted in Figure 4a. Assume, we have two trains, tr_1 with maximal velocity $v_{\text{tr}_1}^{(\max)} = 50 \text{ m/s}$, and tr_2 with maximal velocity $v_{\text{tr}_2}^{(\max)} = 20 \text{ m/s}$. Using $\hat{\tau}(\cdot, \cdot)$ defined in Equation (4), the quickest path of tr_1 from u_0 to u_3 is via u_1 and

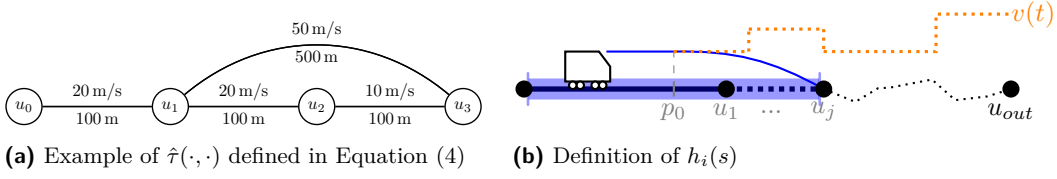
$$\hat{\tau}_1(u_0, u_3) = \frac{100 \text{ m}}{20 \text{ m/s}} + \frac{500 \text{ m}}{50 \text{ m/s}} = 5 \text{ s} + 10 \text{ s} = 15 \text{ s}, \quad (5)$$

whereas the quickest path of tr_2 is via u_1 and u_2 with

$$\hat{\tau}_2(u_0, u_3) = \frac{100 \text{ m}}{20 \text{ m/s}} + \frac{100 \text{ m}}{20 \text{ m/s}} + \frac{100 \text{ m}}{10 \text{ m/s}} = 5 \text{ s} + 5 \text{ s} + 10 \text{ s} = 20 \text{ s}. \quad (6)$$

However, we do not use $\hat{\tau}_i$ directly as a heuristic. Recall that $h_i(s)$ should estimate $g^*(s, \mathcal{T})$ and be optimistic. However, in state s , tr_i might be forced to slow down because it has to stop at the end of its partially defined route. This is not the case in a terminal state $t \in \mathcal{T}$, so the actual travel time difference can be less than $\tau_i^*(s, t)$. However, this can easily

⁸ not necessarily vertices



■ **Figure 4** Heuristic to use within the A* algorithm

be incorporated into the heuristic. Consider Figure 4b and assume that at point p_0 and time t_0 , a train starts to decelerate only because it has to stop at the end of its partial route. While braking, it traverses vertices u_1, u_2, \dots, u_j and finally stops at u_j at time t_j . These values can be easily returned from the (black-box) simulator while determining $g(s)$ (Equation (1)). In a terminal state, tr_i might not be forced to slow down at p_0 , but could potentially follow the orange speed profile in Figure 4b. Because of this, it would reach u_j earlier than t_j in this terminal state, and the final exit time might be less than $t_j + \tau_i^*(u_j, u_{out}^{(\text{tr}_i)})$. We need to adjust for the time that tr_i could reach u_j earlier than t_j in any terminal state reachable from s . This can be achieved by defining

$$h_i(s) := \underbrace{\hat{\tau}_i(p_0, u_1) + \sum_{k=2}^j \hat{\tau}_i(u_{k-1}, u_k) - (t_j - t_0)}_{\leq 0} + \hat{\tau}_i(u_j, u_{out}^{(\text{tr}_i)}). \quad (7)$$

Finally, we combine these individual heuristics into the final h used in Algorithm 1:

$$h(s) := \sum_{i=1}^{\#\text{tr}} w_i \cdot h_i(s). \quad (8)$$

► **Lemma 15.** *h defined in Equation (8) is consistent.*

Proof. W.l.o.g., assume that we use the simple successor state exploration since the multi-edge version can be seen as traversing multiple transitions in one iteration. Let s be any state and $s^+ \in \Gamma(s)$ be any successor state. Then s and s^+ differ by a single action on a single train tr_i . Let $\bar{t}_i^{(s)}$ and $\bar{t}_i^{(s^+)}$ be the times at which tr_i either exits the network or reaches the end of its route in state s and s^+ , respectively. Then

$$c(s, s^+) = g(s^+) - g(s) = w_i \cdot (\bar{t}_i^{(s^+)} - \bar{t}_i^{(s)}) \quad (9)$$

and

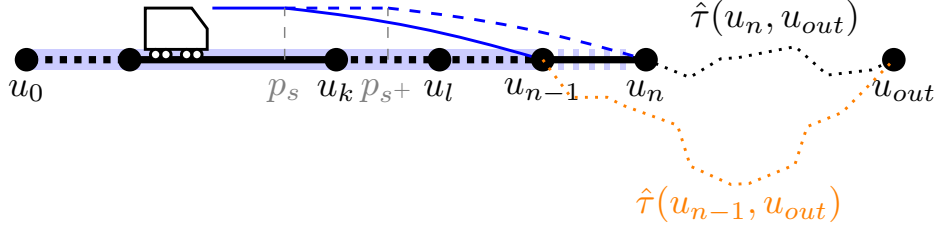
$$h(s) - h(s^+) = w_i \cdot (h_i(s) - h_i(s^+)). \quad (10)$$

If the transition $s \rightarrow s^+$ relates to tr_i stopping in a station, we have

$$h_i(s) - h_i(s^+) = 0 \leq \Delta t = \bar{t}_i^{(s^+)} - \bar{t}_i^{(s)}, \quad (11)$$

where Δt denotes the stopping time in the respective station.

Otherwise, let (u_0, u_1, \dots, u_n) be the path of tr_i in state s^+ . Hence, (u_0, \dots, u_{n-1}) is its path in state s . Let p_s be the braking point in state s and p_{s^+} in state s^+ respectively, see Figure 5. Let u_k ($0 < k \leq n-1$) be the first vertex after p_s and u_l ($k \leq l \leq n$) be the first vertex after p_{s^+} . Then



■ **Figure 5** Proof that h defined in Equation (8) is consistent

$$\begin{aligned}
 h_i(s) - h_i(s^+) &= \hat{\tau}_i(p_s, u_k) + \sum_{j=k+1}^{n-1} \hat{\tau}_i(u_{j-1}, u_j) - (t_{u_{n-1}}^{(s)} - t_{p_s}^{(s)}) + \hat{\tau}_i(u_{n-1}, u_{out}^{(tr_i)}) \\
 &\quad - \hat{\tau}_i(p_{s+}, u_l) - \sum_{j=l+1}^n \hat{\tau}_i(u_{j-1}, u_j) + (t_{u_n}^{(s^+)} - t_{p_{s+}}^{(s^+)}) - \hat{\tau}_i(u_n, u_{out}^{(tr_i)}) \quad (12)
 \end{aligned}$$

By optimality of the quickest path, we obtain

$$\hat{\tau}_i(u_{n-1}, u_{out}^{(tr_i)}) - \hat{\tau}_i(u_n, u_{out}^{(tr_i)}) \leq \underbrace{\hat{\tau}_i(u_{n-1}, u_n) + \hat{\tau}_i(u_n, u_{out}^{(tr_i)}) - \hat{\tau}_i(u_n, u_{out}^{(tr_i)})}_{=0}. \quad (13)$$

Moreover, the adjustments made to h_i due to early braking nicely cancel out to

$$\begin{aligned}
 &\hat{\tau}_i(p_s, u_k) + \sum_{j=k+1}^{n-1} \hat{\tau}_i(u_{j-1}, u_j) - \hat{\tau}_i(p_{s+}, u_l) - \sum_{j=l+1}^{n-1} \hat{\tau}_i(u_{j-1}, u_j) \\
 &= \hat{\tau}_i(p_s, u_k) + \sum_{j=k+1}^{l-1} \hat{\tau}_i(u_{j-1}, u_j) + \hat{\tau}_i(u_{l-1}, u_l) - \hat{\tau}_i(p_{s+}, u_l) \quad (14)
 \end{aligned}$$

$$\leq \hat{\tau}_i(p_s, u_k) + \sum_{j=k+1}^{l-1} \hat{\tau}_i(u_{j-1}, u_j) + \hat{\tau}_i(u_{l-1}, p_{s+}) + \hat{\tau}_i(p_{s+}, u_l) - \hat{\tau}_i(p_{s+}, u_l) \quad (15)$$

$$= \hat{\tau}_i(p_s, u_k) + \sum_{j=k+1}^{l-1} \hat{\tau}_i(u_{j-1}, u_j) + \hat{\tau}_i(u_{l-1}, p_{s+}) \leq (t_{p_{s+}}^{(s^+)} - t_{p_s}^{(s^+)}) \quad (16)$$

assuming $l > k$ (analog if $l = k$). Hence, we obtain

$$\begin{aligned}
 h_i(s) - h_i(s^+) &\leq (t_{p_{s+}}^{(s^+)} - t_{p_s}^{(s^+)}) - (t_{u_{n-1}}^{(s)} - t_{p_s}^{(s)}) + (t_{u_n}^{(s^+)} - t_{p_{s+}}^{(s^+)}) \\
 &\quad + \hat{\tau}_i(u_{n-1}, u_n) - \hat{\tau}_i(u_{n-1}, u_n) \quad (17)
 \end{aligned}$$

$$= (t_{p_{s+}}^{(s^+)} - t_{p_s}^{(s)}) - (t_{u_{n-1}}^{(s)} - t_{p_s}^{(s)}) + (t_{u_n}^{(s^+)} - t_{p_{s+}}^{(s^+)}) \quad (18)$$

$$= (t_{u_n}^{(s^+)} - t_{u_{n-1}}^{(s)}) = \bar{t}_i^{(s^+)} - \bar{t}_i^{(s)}, \quad (19)$$

where we use that $t_{p_s}^{(s^+)} = t_{p_s}^{(s)}$, because the constraint to stop at u_j affects tr_i only after reaching point p_s , hence, the train behavior up to p_s are identical in states s and s^+ . Similarly, one can easily show that $h_i(s) - h_i(s^+) \leq \bar{t}_i^{(s^+)} - \bar{t}_i^{(s)}$ also holds for the edge cases when a train enters or exits the network. Overall

$$h(s) - h(s^+) = w_i \cdot (h_i(s) - h_i(s^+)) \leq w_i \cdot (\bar{t}_i^{(s^+)} - \bar{t}_i^{(s)}) = c(s, s^+) \quad (20)$$

proving that h is consistent. ◀

► **Theorem 16.** *Algorithm 1 together with g defined in Equation (1) and h defined in Equation (8) outputs an optimal solution to Problem 3 under Assumption 7.*

Proof. Follows directly from Lemma 15 and Theorem 6. ◀

4.4 Extending the Heuristic

The heuristic presented in Section 4.3 can be further extended to approximate the remaining time more accurately by using more information provided by Problem 3. Note that the train has to visit a specified list of stations before leaving the network, so it might not be possible for a train to take the quickest route anyway. Assume that in state s , tr_i has not stopped in stations $\mathcal{S}_l, \dots, \mathcal{S}_{m_i}$ yet, i.e., $\hat{m}_i = l - 1$ in Definition 9, then we can update Equation (7) to

$$\begin{aligned} h_i(s) := & \hat{\tau}_i(p_0, u_1) + \sum_{k=2}^j \hat{\tau}_i(u_{k-1}, u_k) - (t_i - t_0) + \sum_{k=l}^{m_i} \Delta t_k^{\text{tr}_i} \\ & + \hat{\tau}_i(u_j, \mathcal{S}_l) + \sum_{k=l}^{m_i-1} \hat{\tau}_i(\mathcal{S}_k, \mathcal{S}_{k+1}) + \hat{\tau}_i(\mathcal{S}_{m_i}, u_{\text{out}}^{(\text{tr}_i)}), \end{aligned} \quad (21)$$

where

$$\hat{\tau}_i(A, B) := \min_{p_1 \in A, p_2 \in B} \hat{\tau}_i(p_1, p_2) \quad (22)$$

is the natural extension of $\hat{\tau}_i$ to distances between sets.

Moreover, Problem 3 provides information on the earliest departure times. Since the train is not allowed to leave before that time, we can already incorporate this into the heuristic. For this, set

$$h_i^{(l)}(s) := \hat{\tau}_i(p_0, u_1) + \sum_{k=2}^j \hat{\tau}_i(u_{k-1}, u_k) - (t_i - t_0) + \hat{\tau}_i(u_j, \mathcal{S}_l) + \Delta t_l^{\text{tr}_i}. \quad (23)$$

For any future station, we can already incorporate the earliest departure times of the previous station by

$$h_i^{(k)}(s) := \max \left\{ h_i^{(k-1)}(s), \underline{\delta}_{k-1}^{\text{tr}_i} \right\} + \hat{\tau}_i(\mathcal{S}_{k-1}, \mathcal{S}_k) + \Delta t_k^{\text{tr}_i} \quad (24)$$

and, finally,

$$h_i(s) := \max \left\{ h_i^{(m_i)}(s), \underline{\delta}_{m_i}^{\text{tr}_i} \right\} + \hat{\tau}_i(\mathcal{S}_{m_i}, u_{\text{out}}^{(\text{tr}_i)}). \quad (25)$$

► **Corollary 17.** *The heuristics h induced from Equations (21) and (25) are consistent and, hence, Algorithm 1 outputs an optimal solution to Problem 3 under Assumption 7.*

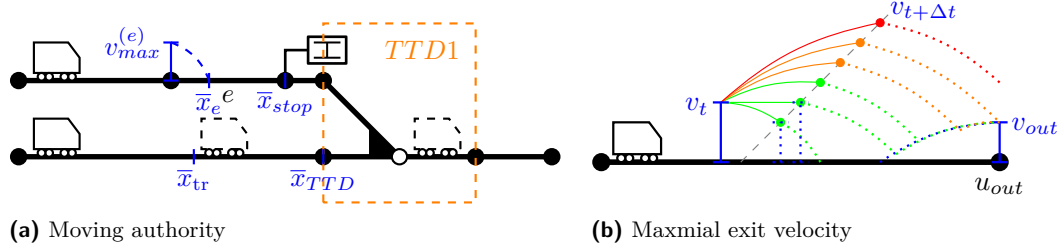
Proof. Analog to the proof of Lemma 15. Equation (11) changes to

$$h_i(s) - h_i(s^+) = \Delta t = \bar{t}_i^{(s^+)} - \bar{t}_i^{(s)}. \quad (26)$$

The main ideas of the proof carry over. ◀

5 Evaluation of Objective Value using Simulation

For implementation, it remains to have access to a simulator for evaluating $g(\cdot)$ (Assumption 11). To showcase the proposed method, we implement a simulator based on Assumption 7 and simple laws of motion, i.e., every train has a constant maximal acceleration $a_{\max}^{(\text{tr})} > 0$ and deceleration $d_{\max}^{(\text{tr})}$. In theory, one could use any simulation tool at hand and, e.g., even incorporate more exact braking curves.



■ **Figure 6** Moving authority and maximal exit velocity

5.1 Principles

For simplicity, we use a time discretization of $\Delta t := 6$ s, which seems to be the standard time between two consecutive train position reports [3, Section 7.5.1.143]. For every time step $t \in \{0, 6, \dots\}$, we keep track of the velocity $v_t^{(tr)} \in [0, v_{max}^{tr}]$ and position⁹ $x_t^{(tr)}$ of each train. In between, we assume linear movement; hence,

$$x_{t+\Delta t}^{(tr)} - x_t^{(tr)} = \frac{v_{t+\Delta t}^{(tr)} + v_t^{(tr)}}{2} \cdot \Delta t. \quad (27)$$

Assume that at time t , the signaling system allows a train to move at most a distance $\bar{x}_t^{(tr)}$ (moving authority). Using basic laws of motion, the braking distance of a train at speed v is given by $\frac{v^2}{2 \cdot d_{max}^{(tr)}}$, hence,

$$\frac{v_{t+\Delta t}^{(tr)} + v_t^{(tr)}}{2} \cdot \Delta t + \frac{\left(v_{t+\Delta t}^{(tr)}\right)^2}{2 \cdot d_{max}^{(tr)}} \leq \bar{x}_t^{(tr)}, \quad (28)$$

and, thus,

$$v_{t+\Delta t}^{(tr)} \leq \frac{1}{2} \cdot \left(\sqrt{(d \cdot \Delta t)^2 + 4 \cdot (2 \cdot d \cdot \bar{x}_t^{(tr)} - d \cdot \Delta t \cdot v_t^{(tr)})} - d \cdot \Delta t \right) =: \nu \left(v_t^{(tr)}, \bar{x}_t^{(tr)} \right). \quad (29)$$

In general, we will set $v_{t+\Delta t}^{(tr)} = \nu \left(v_t^{(tr)}, \bar{x}_t^{(tr)} \right)$ by Assumption 7 unless there are further constraints on the maximal velocity (see Section 5.3).

5.2 Moving Authority

There are four main reasons why the moving authority may be restricted. They are exemplarily depicted in Figure 6a. A train may only advance to the rear of a preceding train, its next scheduled stop position, or the beginning of a TTD section, which the preceding train has not fully cleared yet. In order to enforce speed constraints on future edges (that are not reachable within one timestep¹⁰), we restrict the moving authority by the point at which the train would stop if it decelerates at full rate just after entering the edge at its speed limit.

⁹ The position refers to the front of the train. The rear end can be directly induced from its length.

¹⁰ refer to Section 5.3 in that case

5.3 Restrictions on Maximal Velocity

Finally, there are some cases where it is more natural to restrict the speed directly instead of incorporating it into the moving authority, in which case we might not be able to set $v_{t+\Delta t}^{(\text{tr})} = \nu \left(v_t^{(\text{tr})}, \bar{x}_t^{(\text{tr})} \right)$. Naturally, the speed is restricted by the speed limit of the current edge (and any edge that can be reached within one timestep) as well as the train’s maximal velocity. Moreover, by maximal acceleration, $v_{t+\Delta t}^{(\text{tr})} \leq v_t^{(\text{tr})} + a \cdot \Delta t$. The most complex restriction is to ensure that a train can leave the network at its target velocity, but only at a time $\geq \underline{t}_{\text{out}}^{(\text{tr})}$. In that case, a train might need to slow down in order not to arrive at the exit too early but still have enough distance left to accelerate to the target exit velocity $v_{\text{out}}^{(\text{tr})}$ before leaving the network. Assume that $v_{t+\Delta t}^{(\text{tr})}$ is fixed, then it is easy to calculate the maximal possible runtime to the exit node by assuming that the train decelerates until the point where it has to accelerate again to reach the target velocity at exit. Trying difference values for $v_{t+\Delta t}^{(\text{tr})}$, they can either lead to an exit at time $\geq \underline{t}_{\text{out}}^{(\text{tr})}$ (green lines in Figure 6b), at time $< \underline{t}_{\text{out}}^{(\text{tr})}$ (orange lines) or the target velocity cannot be attained at exit (red lines). Using binary search, we choose the largest value for $v_{t+\Delta t}^{(\text{tr})}$ that still leads to an exit at time $\geq \underline{t}_{\text{out}}^{(\text{tr})}$.

6 Case Study

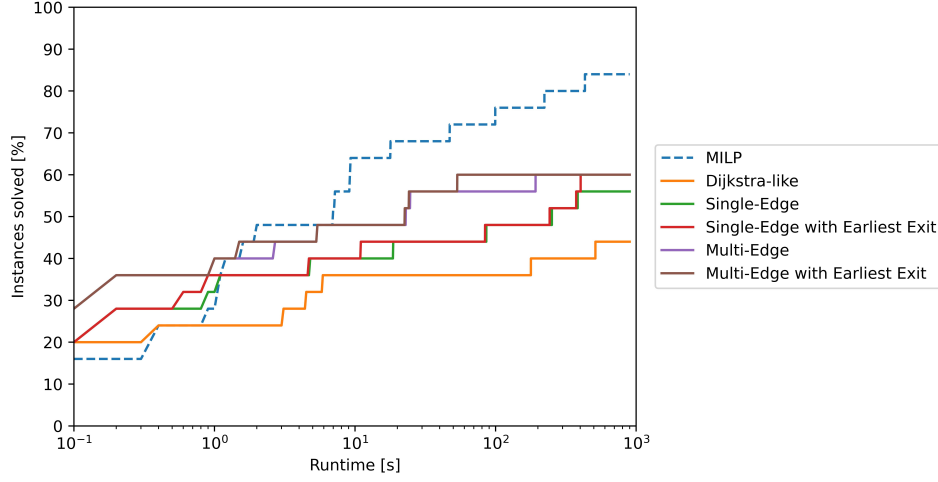
To test the proposed method, we implement Algorithm 1 together with the transitions described in Section 4.2, the heuristics in Section 4.4, and the simulator in Section 5. All code is available open-source as part of the *Munich Train Control Toolkit* (MTCT) available on GitHub at <https://github.com/cda-tum/mtct>. The user can choose among different strategies when executing the A* algorithm.

- *Dijkstra-like*: This setting makes use of the single-edge transitions described in Section 4.2 and does not use a heuristic, i.e., $h(s) \approx 0$. However, in order to remain consistent, the heuristic amends for the time “lost” by braking at the end of its partial route, i.e., $h_i(s) := \hat{\tau}_i(p_0, u_1) + \sum_{k=2}^j \hat{\tau}_i(u_{k-1}, u_k) - (t_i - t_0) \leq 0$.
- *Single-Edge*: This setting combines the single-edge transitions described in Section 4.2 together with the heuristic defined in Equation (21), i.e., future stations are considered in the approximation, however not the earliest exit times defined in the problem instance.
- *Single-Edge with Earliest Exit*: Same as “Single-Edge”, but using Equation (25) for the heuristic, instead, i.e., also considering minimal runtimes due to the constraints on earliest departure in the problem description.
- *Multi-Edge* and *Multi-Edge with Earliest Exit*: Same as above, but using the multi-edge transitions described in Section 4.2

We tested these strategies on an Intel(R) Xeon(R) W-1370P system using a 3.60GHz CPU (8 cores) and 128GB RAM running Ubuntu 20.04. We use the benchmark network from [5]. Additionally, we use the random timetables generated by [6] on two of the networks, including Munich’s S-Bahn Stammstrecke. We compare runtimes to the MILP-approach [6].

The results are plotted in Figure 7, and raw data is provided in Appendix B. On the x-axis, we denote the runtime in seconds¹¹. The y-axis corresponds to the fraction of instances solved within that time (or faster). Hence, a line to the left/top is generally better. By design, all lines are monotonically increasing, and the left ends of each “step” are the runtimes of

¹¹ Note that the scale is logarithmic.



■ **Figure 7** Runtime comparison

some instance. From the horizontal distance between the lines, one can approximately¹² read off the runtime difference on that instance.

We can see a clear trend. Reducing the size of the explored solution space by skipping states with only one sensible successor significantly reduces the runtime. Additionally, using as much information as possible already in the heuristic is beneficial, i.e., the one induced by Equation (25). This trend suggests that there is still significant performance potential if both transitions and guiding heuristics are improved. Instead of using Equation (4) for runtime approximation, one could, e.g., use the minimal runtimes in [14], incorporating simple acceleration and deceleration limitations.

7 Conclusions

In this work, we have proposed an algorithm based on A* (Sections 3–5) that can find optimal train routings on networks equipped with moving block control systems (Problem 3) under a reasonable assumption on driver behavior (Assumption 7) and showed its applicability to benchmark instances (Section 6). It is designed in such a way that any arbitrary black-box function can be used to evaluate the arising states. Because of this, our algorithm can, e.g., be used with any arbitrary detailed simulation tool that might consider more detailed braking curves or headways than reasonable to model within a MILP framework.

It is not unexpected that the increased accuracy (while still guaranteed to be optimal) comes with a downside in runtime. At the same time, Section 6 shows the general applicability of our approach and how the choice of the guiding heuristic and transition strategy affects the overall runtime and number of solvable instances. By simple extensions to these, we were able to improve the efficiency notably, even if the runtime of previous MILP approaches could not be reached yet on larger instances.

At the same time, our approach is model-agnostic and flexible. It is not limited to being used within an (exact) A* method. Instead, any search algorithm can be applied.

¹² assuming that the order of instances solved is identical for every algorithm, which is, of course, not guaranteed

In particular, approximative approaches might have the potential for further significant performance gains. The simplest choice might be a weighted version of A* that guides the search quicker towards terminal states [13]. However, we are not limited to variants of A*, but one could, e.g., also consider genetic algorithms, local search, reinforcement learning, and more.

Overall, this work provides a valuable basis for applying general and well-established search algorithms to routing tasks on railway networks. The modeling can be arbitrarily exact by using any (possibly black-box) evaluation methods, such as simulation. Research on improved guiding of the search algorithms and exploring the potential of approximative search algorithms on the presented encoding is left to future work.

References

- 1 Ralf Borndörfer, Torsten Klug, Leonardo Lamorgese, Carlo Mannino, Markus Reuther, and Thomas Schlechte, editors. *Handbook of Optimization in the Railway Industry*. Springer International Publishing, 2018. URL: <https://doi.org/10.1007/978-3-319-72153-8>.
- 2 Stefan Edelkamp. *Heuristic search*. Morgan Kaufmann, Waltham, MA, 2012. URL: <https://doi.org/10.1016/C2009-0-16511-X>.
- 3 EEIG ERTMS USERS GROUP. *ERTMS/ETCS System Requirements Specification, SUBSET-026*. European Union Agency for Railways, 2023.
- 4 Stefan Engels, Tom Peham, Judith Przigoda, Nils Przigoda, and Robert Wille. Design tasks and their complexity for the European Train Control System with hybrid train detection. *EURO Journal on Transportation and Logistics*, 14:100161, 2025. URL: <https://doi.org/10.1016/j.ejtl.2025.100161>.
- 5 Stefan Engels, Tom Peham, and Robert Wille. A symbolic design method for ETCS Hybrid Level 3 at different degrees of accuracy. In Daniele Frigioni and Philine Schiewe, editors, *23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, volume 115 of *OASICS*, pages 6:1–6:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: <https://doi.org/10.4230/OASICS.ATMOS.2023.6>.
- 6 Stefan Engels and Robert Wille. Comparing lazy constraint selection strategies in train routing with moving block control. In Marek Bolanowski, Maria Ganzha, Leszek A. Maciaszek, Marcin Paprzycki, and Dominik Slezak, editors, *Proceedings of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS)*, volume 39 of *Annals of Computer Science and Information Systems*, pages 585–590. Polish Information Processing Society, 2024. URL: <https://doi.org/10.15439/2024F3041>.
- 7 Stefan Engels and Robert Wille. Towards an optimization pipeline for the design of train control systems with hybrid train detection (short paper). In Paul C. Bouman and Spyros C. Kontogiannis, editors, *24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, volume 123 of *OASICS*, pages 12:1–12:6. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: <https://doi.org/10.4230/OASICS.ATMOS.2024.12>.
- 8 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968. URL: <https://doi.org/10.1109/TSSC.1968.300136>.
- 9 Simon Hofmann, Marcel Walter, and Robert Wille. A* is born: Efficient and scalable physical design for field-coupled nanocomputing. In *2024 IEEE 24th International Conference on Nanotechnology (NANO)*, pages 80–85. IEEE, 2024. URL: <https://doi.org/10.1109/nano61778.2024.10628808>.
- 10 Torsten Klug, Markus Reuther, and Thomas Schlechte. Does laziness pay off? - A lazy-constraint approach to timetabling. In Mattia D’Emidio and Niels Lindner, editors, *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and*

- Systems (ATMOS)*, volume 106 of *OASICS*, pages 11:1–11:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: <https://doi.org/10.4230/OASICS.ATMOS.2022.11>.
- 11 Jörn Pachl. *Railway Signalling Principles: Edition 2.0*. Universitätsbibliothek Braunschweig, 2021. URL: <https://doi.org/10.24355/dbbs.084-202110181429-0>.
 - 12 Tom Peham, Judith Przigoda, Nils Przigoda, and Robert Wille. Optimal railway routing using virtual subsections. In Simon Collart Dutilleul, Anne E. Haxthausen, and Thierry Lecomte, editors, *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification (RSSRail)*, volume 13294 of *Lecture Notes in Computer Science*, pages 63–79. Springer International Publishing, 2022. URL: https://doi.org/10.1007/978-3-031-05814-1_5.
 - 13 Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3):193–204, 1970. URL: [https://doi.org/10.1016/0004-3702\(70\)90007-X](https://doi.org/10.1016/0004-3702(70)90007-X).
 - 14 Thomas Schlechte, Ralf Borndörfer, Jonas Denßen, Simon Heller, Torsten Klug, Michael Küpper, Niels Lindner, Markus Reuther, Andreas Söhlke, and William Steadman. Timetable optimization for a moving block system. *Journal of Rail Transport Planning & Management*, 22:100315, 2022. URL: <https://doi.org/10.1016/j.jrtpm.2022.100315>.
 - 15 Lars Schnieder. *Communications-Based Train Control (CBTC)*. Springer Berlin Heidelberg, 2021. URL: <https://doi.org/10.1007/978-3-662-62876-8>.

A

 Railway Network

In this paper, we use the formal model of a railway network introduced in [4]. In general, a railway network is given as a directed graph, i.e., there might be restrictions on travel direction. If $(u, v) \in E$ and $(v, u) \in E$, we sometimes use undirected edges for illustration purposes. Every edge $e \in E$ has a specified length $\text{len}(e) \in \mathbb{R}_{>0}$ as well as a maximal speed $v_e^{(max)} \in \mathbb{R}_{>0}$. Because turnouts (i.e., vertices $v \in V$ with $\deg(v) \geq 3$) do not allow arbitrary transitions in general, we are given a successor function $s_v: \delta^{in}(v) \rightarrow \mathcal{P}(\delta^{out}(v))$ for every vertex. If $e^+ \in s_v(e)$, the railway network allows a train to move from edge e to e^+ via v . Moreover, a set of border vertices $\mathcal{B} \subseteq V$ is specified at which trains can enter and leave the railway network with predefined headway times. Finally, even though we consider moving block control systems, some TTD sections with classical train separation are given to model basic flank protection around turnouts.

- **Definition 18** (Railway Network). A railway network $N = (G, \text{len}, \{s_v\}_{v \in V})$ is defined by
- a directed graph $G = (V, E)$ with vertices V being the set of points of interest and edges E being the railway tracks between the aforementioned points of interest,
 - a mapping $\text{len}: E \rightarrow \mathbb{R}_{>0}$ denoting the length of each edge such that $\text{len}(e) = \text{len}(e^\circ)$ for every pair of edges $e, e^\circ \in E^{13}$,
 - maximal velocities $v_e^{(max)} \in \mathbb{R}_{>0}$ for every $e \in E$,
 - a family of mappings $\{s_v\}_{v \in V}$, where $s_v: \delta^{in}(v) \rightarrow \mathcal{P}(\delta^{out}(v))$ represents the valid movements over v , and
 - border vertices $\mathcal{B} \subseteq V$ together with headway times $h: \mathcal{B} \rightarrow \mathbb{R}_{\geq 0}$.

We refer to [4] for more details and examples.

¹³For $e = (u, v) \in E$, $e^\circ := (v, u)$

B

 Raw Data

Instance	MILP t[s]	Dijkstra-like t[s] #it	Single-Edge		SE with Earliest Exit		Multi-Edge		ME with Earliest Exit	
			t[s]	#it	t[s]	#it	t[s]	#	t[s]	#it
High Speed Track (02 Trains)	0.03	0.00 4	0.00	4	0.00	4	0.00	4	0.00	4
High Speed Track (05 Trains)	0.02	0.07 177	0.01	32	0.01	26	0.01	32	0.01	26
Overtake	timeout	4.43 36141	1.02	8267	0.54	4700	0.19	1091	0.10	597
Simple 2-Track Station	0.40	0.39 4586	0.15	1872	0.11	1364	0.06	449	0.04	319
Simple Network	1.56	178.30 357054	18.70	37810	10.99	25890	2.61	4796	1.46	2971
Simple Network (03 Random Trains)	1.20	5.85 24065	0.83	3706	0.83	3706	0.17	625	0.17	625
Simple Network (06 Random Trains)	9.21	1622725	timeout	1333013	402.94	376782	192.57	153066	53.34	45160
Simple Network (09 Random Trains)	9.18	1420517	timeout	1318516	timeout	1216819	timeout	853380	timeout	791987
Simple Network (12 Random Trains)	17.82	802635	timeout	874964	timeout	814870	timeout	498327	timeout	495954
Simple Network (15 Random Trains)	99.43	606614	timeout	536984	timeout	573502	timeout	396691	timeout	386992
Simple Network (18 Random Trains)	47.15	433182	timeout	362013	timeout	356877	timeout	262670	timeout	264495
Simple Network (21 Random Trains)	timeout	302300	timeout	241350	timeout	290107	timeout	185869	timeout	182279
Simple Network (24 Random Trains)	timeout	209872	timeout	176201	timeout	179720	timeout	114616	timeout	100410
Simple Network (27 Random Trains)	222.51	177386	timeout	150395	timeout	145467	timeout	105765	timeout	98356
Simple Network (30 Random Trains)	432.38	126364	timeout	99878	timeout	109218	timeout	72243	timeout	72950
Single Track With Station	0.04	0.01 114	0.00	44	0.00	39	0.00	32	0.00	28
Single Track Without Station	0.05	0.00 4	0.00	4	0.00	4	0.00	4	0.00	4
Stammstrecke (01 Random Trains)	0.35	0.00 57	0.00	23	0.00	23	0.00	9	0.00	9
Stammstrecke (02 Random Trains)	1.03	3.08 30576	0.18	1221	0.18	1221	0.07	289	0.07	289
Stammstrecke (03 Random Trains)	1.04	513.15 2695389	4.80	19278	4.67	19278	0.97	2392	0.94	2392
Stammstrecke (04 Random Trains)	timeout	7748178	85.95	245384	84.14	245384	5.37	10642	5.31	10642
Stammstrecke (04 Trains)	0.89	8110539	251.52	501466	242.45	501466	24.66	31872	24.16	31872
Stammstrecke (05 Random Trains)	7.11	5186928	383.87	817560	374.54	817560	22.94	35648	22.57	35648
Stammstrecke (08 Trains)	1.94	3758144	timeout	2528215	timeout	958400	timeout	1277847	timeout	572097
Stammstrecke (16 Trains)	6.91	1521604	timeout	748951	timeout	230370	timeout	597978	timeout	142532