

Time-Aware 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

Railway networks are operating at their capacity limits. Modern control systems allow for shorter train-following times on existing infrastructure, resulting in faster travel times and higher capacity. As new degrees of freedom emerge, developing effective and flexible algorithms becomes essential to incorporate these advancements into the planning process. Previously, a flexible method for generating provably optimal timetables using A* search has been proposed. However, in practice, the algorithm spends significant time exploring states that yield no new information. This work addresses this issue and shows that transforming the state space into a (directed) tree drastically reduces runtime and enhances scalability, particularly with respect to the number of trains. The implementation is available open-source as part of the *Munich Train Control Toolkit* on GitHub at <https://github.com/cda-tum/mtct>.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases ETCS, European Train Control System, Train Routing, Moving Block, A*, Search Space Reduction, Munich Train Control Toolkit

Digital Object Identifier 10.4230/OASICS.ATMOS.2026.3

Supplementary Material *Software (Source Code)*: <https://github.com/cda-tum/mtct> [2]
archived at `swb:1:dir:2ffb77c8877a842cbc78f45016d4270dd5aeff5d`

1 Introduction

Sustainable transportation relies heavily on railway transportation. At the same time, railway networks are already operating at their capacity limits.

In general, trains cannot operate on sight due to their long braking distances. Hence, control systems have to be implemented to prevent collisions. Previous national rules have been unified across Europe and other regions, most notably, the *European Train Control System* (ETCS, Europe and outside of Europe), *Chinese Train Control System* (CTCS, China), *Positive Train Control System* (PTC, North America), and *Communication Based Train Control* (CBTC, metro systems) [8, 12].

These control systems come in different variants. Modern versions (e.g., moving-block-based) allow for shorter train-following times and, thus, increased capacity. At the same time, train separation is no longer enforced with discrete blocks or even at discrete points, but is continuously monitored.

Since the separation imposed by the control system significantly affects the feasible trajectories of trains, they need to be considered when, e.g., creating timetables. Most research on optimization problems within that context focuses on classical control systems [1]. We are only aware of limited research on modern moving-block systems for timetable optimization and train routing [11, 7, 3, 4].

¹ Corresponding author



© Stefan Engels and Robert Wille;
licensed under Creative Commons License CC-BY 4.0

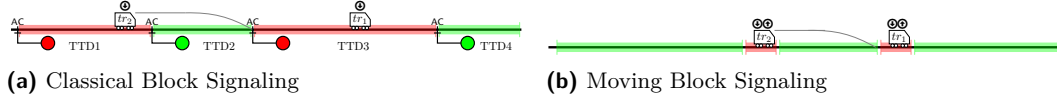
26th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2026).

Editors: Valentina Cacchiani and Stefan Funke; Article No. 3; pp. 3:1–3:19

OpenAccess Series in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Schematic drawings of different signaling principles [3].

[4] proposes a flexible approach based on search algorithms such as A*, which have already been successfully applied to various design automation tasks, e.g., in nanotechnologies [6] and even within the railway domain [9]. Their approach can use any (possibly black-box) simulator to evaluate given states. By doing so, constraints enforced by the control systems, which are usually hard to model, do not have to be discretized, simplified, and/or approximated.

Even though [4] proposes various strategies to improve runtime, it cannot solve several instances on the benchmark it tested. This is partially due to the search algorithm revisiting many states it has already explored, so it does not gain any new information. In this work, we propose adaptations to the search space that prevent such reexplorations, thereby drastically reducing the search space. The key idea is to make the state expansion time-aware, so that the search focuses only on the earliest pending routing decision. This reduces redundant decisions that do not provide new information, while retaining the optimality guarantee. A case study confirms significant runtime improvements on a benchmark set.

The remainder of this work is structured as follows. Section 2 reviews relevant background on railway systems and search algorithms. Section 3 describes the previous approach by [4], focusing on the details relevant to our proposed changes and explaining what structural properties limit its scalability. Based on this, Section 4 proposes further reductions of the search space and is the main contribution of this work. Finally, a case study (Section 5) demonstrates significant improvements to the scalability with respect to the number of trains. Section 6 concludes this work.

2 Background

This section provides basic concepts and definitions relevant to this work.

2.1 Train Control Principles

Because trains usually have long braking distances, control systems are required to prevent collisions. Classically, such systems rely on predefined fixed blocks. Because those blocks often require *trackside train detection* (TTD) systems, we refer to them as TTD sections. As long as a train is anywhere within a block section, any other train has to be able to come to a full stop before reaching that block section, even if the boundary of the section might be far away from the actual position of the blocking train.

Modern control systems, on the other hand, no longer rely on such fixed sections. Instead, trains are continuously separated by their absolute braking distances and trains “only” have to be able to stop before the previous train’s exact position. Generally, this allows shorter train following times.

► **Example 1.** Consider the network in Figure 1. Classically, it might be separated into 4 TTD sections. In Figure 1a, tr_2 is only allowed to advance to the end of TTD2, since TTD3 is occupied. By contrast, in Figure 1b, tr_2 is allowed to advance all the way to tr_1 (possibly minus some buffer).

2.2 Problem Formulation

In this work, we consider the task of routing trains through a part of a railway network. Trains are released at prespecified times at the boundaries of the considered region. They have certain demands, i.e., which stations they have to service. However, the precise route and timing are not yet decided. More precisely, we solve the following routing task:

► **Problem 2** (Optimal Train Routing (on Moving Block Systems)). Given:

- A railway network with vertices V and (directed) edges E as described in [4, Appx. A].
- A set $\mathcal{T}$ of trains, where w_i denotes the weight of importance of train $i \in \mathcal{T}$
- Demands for every train $i \in \mathcal{T}$ consisting of:
 - entry node $u_{in}^{(i)} \in V$ together with entry time $\tau_{in}^{(i)}$ and initial velocity $v_0^{(i)}$,
 - exit node $u_{out}^{(i)} \in V$ together with earliest exit time $\tau_{out}^{(i)}$ and maximal exit velocity $v_{out}^{(i)}$,
 - list of stations $\mathcal{S}_1, \dots, \mathcal{S}_{m_i} \subset E$ together with stopping requests, i.e., a (desired) arrival time $\tau_j^{(i)}$ and a minimal stopping time $\Delta\tau_j^{(i)}$ for every station $\mathcal{S}_j$.

Task: Find a routing satisfying every train's demand, obeying the constraints imposed by a moving-block control system and earliest departure times, and minimizing the (weighted) exit times and average station delays. To trade-off station and exit delays, we add a constant $\alpha \geq 0$. More precisely, let $t_{out}^{(i)}$ be the exit time and $t_1^{(i)}, \dots, t_{m_i}^{(i)}$ the station arrival times of a given train $i \in \mathcal{T}$. The objective (to be minimized) is given by

$$\sum_{i \in \mathcal{T}} \left(w_i \cdot [t_{out}^{(i)} - \tau_{out}^{(i)}]_+ + \alpha \cdot \frac{w_i}{m_i} \sum_{j=1}^{m_i} [t_j^{(i)} - \tau_j^{(i)}]_+ \right), \quad (1)$$

where $[x]_+ := \max(0, x)$ denotes the ReLU function.

Since we assume that a train is not allowed to leave the network before its earliest exit time, Equation (1) is equal to

$$\sum_{i \in \mathcal{T}} \left(w_i \cdot t_{out}^{(i)} + \alpha \cdot \frac{w_i}{m_i} \sum_{j=1}^{m_i} [t_j^{(i)} - \tau_j^{(i)}]_+ \right) - \sum_{i \in \mathcal{T}} w_i \tau_{out}^{(i)}. \quad (2)$$

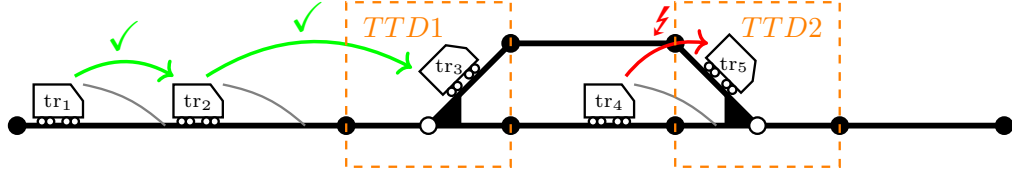
Since $\sum_{i \in \mathcal{T}} w_i \tau_{out}^{(i)}$ is a constant, minimizing Equation (1) is equivalent to minimizing

$$\sum_{i \in \mathcal{T}} \left(w_i \cdot t_{out}^{(i)} + \alpha \cdot \frac{w_i}{m_i} \sum_{j=1}^{m_i} [t_j^{(i)} - \tau_j^{(i)}]_+ \right). \quad (3)$$

For the remainder of the paper, we use the objective in the form presented in Equation (3).

All constraints imposed by the train control system (in our context, a moving-block control system) must be followed. Even though we optimize in the context of a moving-block system, some TTD sections remain on the network. For simplicity and to provide a basic level of flank protection, we assume that all turnouts are surrounded by a classical TTD section. On the contrary, there are usually no other TTD sections in our instances. For the remainder of this work, we can think of TTD sections and turnouts to be somewhat “equivalent”.

► **Example 3.** Consider the network depicted in Figure 2 with 5 trains and relevant braking distances marked with gray lines. tr_1 and tr_2 are on the same track segment. However, this is fine because a moving-block control system is assumed. We only enforce classical TTD



■ **Figure 2** Example network.

separation at the turnouts. Because of that, tr_4 conflicts with tr_5 , because it cannot stop before the TTD section², which tr_5 already entered. On the other hand, tr_2 and tr_3 are safely separated even though a TTD section is enforced around the left turnout.

2.3 (Weighted) A* Algorithm

Let $G = (S, A, c)$ be a directed and weighted graph with vertices S , arcs A , and cost $c: A \rightarrow \mathbb{R}$. Furthermore, we are given an initial vertex $s_0 \in S$ and a goal set $\mathcal{G} \subseteq S$. The goal is to find a minimal cost path from s_0 to any vertex in $\mathcal{G}$. One can also think of the vertices as states, and the arcs as denoting possible actions or transitions with respective costs c .

A* can be used to find such minimal paths. It functions very similarly to Dijkstra; however, instead of expanding the path with the lowest cost, it also adds a guiding heuristic estimating the distance to a goal state, see Algorithm 1, where g denotes the currently best known path cost from s_0 to s and f the cost adjusted with the guiding heuristic.

■ **Algorithm 1** (Weighted) A*-Algorithm [5, 10].

```

C ← ∅; f(s) ← ∞ for all s ∈ V; g(s) ← ∞ for all s ∈ V
add s0 to priority queue (pq); f(s0) ← 0
while pq is not empty:
    select s ∈ argminσ ∈ pq f(σ)
    C ← C ∪ {s}
    if s ∈ G: return s together with observed path
    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+) + ε · h(s+)}
        g(s+) ← min{g(s+), g(s) + c(s, s+)}
return infeasible

```

Special focus should be placed on the choice of the guiding heuristic. If one chooses $h \equiv 0$, Algorithm 1 is basically equivalent to Dijkstra. If h never overestimates the cost of any arc (more precisely, if h is consistent), A* is known to be optimal.

► **Definition 4** (Consistent Heuristic [5]). *A heuristic h is consistent if $h(s_g) = 0$ for every $s_g \in \mathcal{G}$ and $h(s) - h(s^+) \leq c(s, s^+)$ for every $(s, s^+) \in A$.*

The heuristic can be multiplied by a factor $\varepsilon \geq 1$ to get an ε -approximation algorithm. In particular, the following properties and optimality guarantees are well-known:

² If tr_4 were to already be stopped, there would be no conflict.

► **Theorem 5** (ε -Approximation [10]). *If h is consistent, then Algorithm 1 returns a path whose length is at most ε times the minimal path length.*

► **Corollary 6** (Optimality [5]). *If $\varepsilon = 1$ and h is consistent, then Algorithm 1 returns a minimal path.*

3 Previous Work

Previously, an algorithm to solve (a slightly different variant of) Problem 2 has been proposed and implemented [4]. The main idea was that, under the arguably reasonable assumption that trains drive as fast as possible and allowed by the control system, a solution is uniquely defined by the following decisions:

- routing, i.e., the edges used by every train,
- stops, i.e., the exact stop positions within the scheduled stations, and
- ordering, i.e., the order of the trains on every turnout as well as entry and exit vertices.

In that case, any simulator at hand can be used to evaluate a given solution, and A* can be used as a heuristic search algorithm to find an optimal solution.

We summarize the key concepts proposed by [4] in the following subsections. Where applicable, we slightly modify their results to be consistent with this paper.

3.1 State Space

The main idea was to expand routes incrementally. For this, the state space consists not only of all valid solutions, but also of those that are only partially defined and then extended towards fully specified solutions.

3.1.1 States

► **Definition 7** (State [4, Definition 9]). *A state is given by any partial solution. More precisely, it consists of information for every train $i \in \mathcal{T}$:*

- a valid path $e_1^{(i)}, \dots, e_{k_i}^{(i)} \in E$, possibly empty, starting in $u_{in}^{(i)}$,
- a list of positions $s_1^{(i)}, \dots, s_{\hat{m}_i}^{(i)}$ being within the first $0 \leq \hat{m}_i \leq m_i$ scheduled stations $\mathcal{S}_1^{(i)}, \dots, \mathcal{S}_{\hat{m}_i}^{(i)}$, and
- the train ordering on every TTD section as well as on entry and exit vertices.

By design, states do not need to specify paths leading to an exit vertex. If a train's path is not fully specified, it is assumed to travel all the way to its last specified edge and then disappear. At the same time, a train that disappeared within the network blocks its last vertex indefinitely, so that it cannot be overtaken by any other train. Because a train's route might not be fully specified, a partial solution is also not required to stop in all scheduled stations.

► **Assumption 8** (Deterministic State Value). *It is assumed that train movements are uniquely defined by the information contained in Definition 7. This can, e.g., be achieved by the discrete-time simulator proposed in [4, Section 5]; however, any other (black-box) simulator could be used.*

The objective value is then given by

$$g(s) := \sum_{i \in \mathcal{T}} \left(\underbrace{w_i \cdot \hat{t}_{k_i}^{(i)} + \alpha \cdot \frac{w_i}{m_i} \sum_{j=1}^{\hat{m}_i} [\hat{t}_j^{(i)} - \tau_j^{(i)}]_+}_{=: g^{(i)}(s)} \right) = \sum_{i \in \mathcal{T}} g^{(i)}(s), \quad (4)$$

where $\hat{t}_1^{(i)}, \dots, \hat{t}_{\hat{m}_i}^{(i)}$ are the arrival times at the specified stops and $\hat{t}_{k_i}^{(i)}$ is the arrival time at the end of edge $e_{k_i}^{(i)}$. Note that the average stop delay already uses the total number of scheduled stops, even if not all stops are specified yet. This simplifies the definition of a consistent heuristic.

If a state is infeasible (for example, because two trains are blocking each other and cannot reach their final edge), we set $g(s) = \infty$.

► **Definition 9 (Goal States).** The set of goal states $\mathcal{G}$ consists of all states which are fully specified (i.e., for all $i \in \mathcal{T}$, $e_{k_i}^{(i)}$ ends at exit $u_{out}^{(i)}$ and $\hat{m}_i = m_i$) and feasible (i.e., $g(s) < \infty$).

3.1.2 Transitions

Having defined the states, it is necessary to define how the search algorithm can traverse the state space. Given any state s , a successor s^+ is obtained by extending the train path of one train by one or multiple edges. For a specific train, [4] proposes different strategies for choosing the number of edges to extend. One strategy extends by a single edge at a time; another strategy combines edges up to the next turnout or scheduled stop.

However, more important for the remainder of this work is to understand how the train ordering is deduced. For this, [4] assumes that whichever train chooses a certain edge first also travels on that edge first. Since a state is generally expanded on every train, all possible train orderings are eventually expanded.

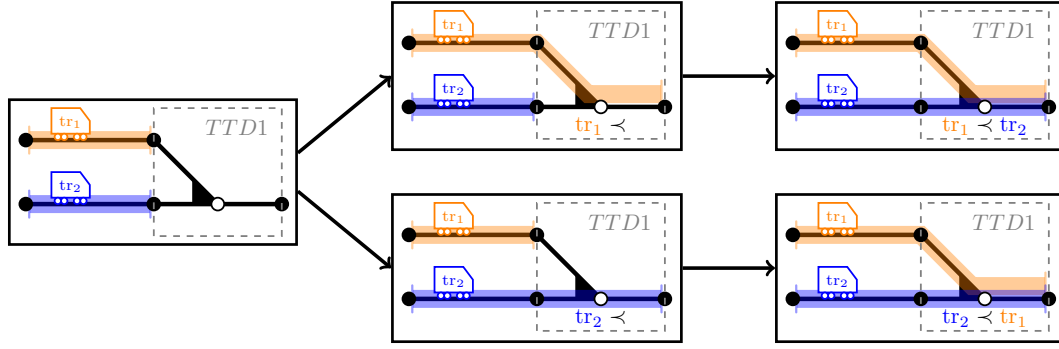
► **Example 10.** Consider Figure 3 with two trains approaching a turnout from different tracks. The left state can be expanded in two ways. One successor state is achieved by moving tr_1 (middle top). By doing so, it is implied that tr_1 moves across the turnout first. This state itself can be expanded by moving tr_2 , in which case the resulting state (top right) assumes that tr_2 waits until tr_1 clears the turnout and then follows. On the other hand, the left state is also expanded by moving tr_2 . Analogously, the bottom right state is reached, in which tr_1 waits for tr_2 to clear the turnout before following.

3.1.3 Cost

Note that we define a constant cost on states only. Thus, the state transition costs used in Algorithm 1 are only given implicitly by $c(s, s^+) := g(s^+) - g(s)$.

W.l.o.g., assume that a single train's path is extended by a certain number of edges. Then every train should either travel identically to the previous state or be slower, because the path extension might lead to further constraints that were not present in the previous state. This motivates the following assumption, which [4] implicitly relies on:

► **Assumption 11 (Monotonicity).** Let s^+ be a state that is created by extending state s , i.e., by adding further edges and/or stops to any of the trains in s . Then $g^{(i)}(s^+) \geq g^{(i)}(s)$ for all $i \in \mathcal{T}$. In particular, $g(s^+) \geq g(s)$.



■ **Figure 3** Train Ordering.

Note that Assumption 11 is not phrased as a theorem, and the justification is not a rigid proof. In fact, depending on the driver's strategy assumed by the simulator, this might not be guaranteed. For example, the Greedy Train Driver Assumption in [4, Assumption 7] does not theoretically guarantee this for all instances and states. At the same time, this did not pose any problems in our case study, since such cases seem unlikely in practice.

3.2 Heuristic

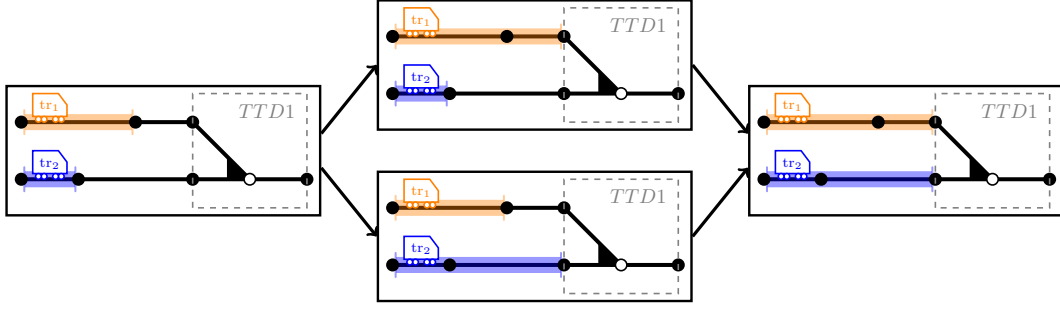
Given any partially defined state, an easy-to-compute guiding heuristic is needed which estimates how far away $g(s)$ is from an optimal goal state, i.e., $h(s) \approx \min_{s_g \in \mathcal{G}} g(s_g) - g(s)$. Obviously, the exact difference is unknown since we are currently searching for the optimal goal state. A natural heuristic is to assume that all trains travel at maximum speed for the remainder of their paths, without any obstacles, i.e., ignoring potential collisions with other trains. [4, Lemma 15] shows that this heuristic is consistent under Assumption 11. Because of this and Corollary 6, they conclude that their proposed method returns optimal solutions to Problem 2 [4, Theorem 16, Corollary 17].

3.3 Problematic State Reexploration

Even though [4] showed that runtime could be improved by reducing the search space using the transition strategies described in Section 3.1.2, the algorithm timed out on many of the tested instances. Adding multiple edges at once and skipping unnecessary states improved the runtime but did not resolve the scalability issue. A major issue with the approach presented in [4] is that many states can be explored multiple times.

► **Example 12.** Consider Figure 4. The right state is expanded from the left state by moving both trains. The algorithm will expand the search space in two ways, reaching the right state. It could either expand tr_1 or tr_2 first. Because both middle states are likely to be considered by the A* search, the state on the right is explored twice. One state was explored without any additional benefit.

Hence, we conclude that it would be desirable to use a tree as the search space, as this would eliminate unnecessary state reexploration. Achieving this is the main contribution of this paper and is described in Section 4.1.



■ **Figure 4** State Reexploration.

4 State Space Reduction

[4] shows that the presented approach using A* search works in principle and finds optimal solutions. However, a naive implementation incurs a high computational cost. To mitigate this, strategies for reducing the search space have been introduced that skip some unnecessary states. This notably improves runtime. At the same time, even the best settings presented in [4] struggle to solve instances with many trains, even if the underlying network structure is rather simple.

This section introduces improvements to this. Recall that generating the set of successor states consists of two steps. First, it is determined which trains are selected³ to advance. Second, for each such train, a set of possible path extensions is generated. We propose improvements for both steps in the following subsections.

4.1 Time-Aware Train Selection

Recall Section 3.3 and that the previous approach by [4] reexplores already expanded states because equivalent decisions are made at several parts of the state space (just in a different order). To prevent this, we aim to make a decision as late as possible. Intuitively, we only want to make the most pressing decision, i.e., the one for the train closest to reaching its specified end-of-route. This makes the train selection somewhat “aware” of the simulated timings of the current state.

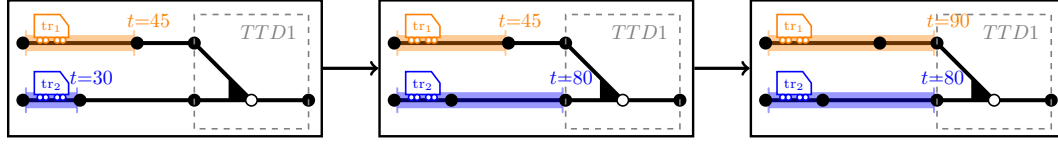
For any given state, we order the trains whose paths are not yet fully specified by the time they reach the end of their currently specified route. We only expand the path of the train that arrives earliest. If multiple trains arrive at the same time, we make a fixed but otherwise arbitrary⁴ choice.

► **Example 13.** Consider Figure 5 with the same situation as in Example 12. Assume that tr_1 needs 45s and tr_2 needs 30s to travel across their first edge. Because the decision for tr_2 is more pressing, it is moved first. Only after that does tr_1 arrive at its route end earliest, so that it is expanded second. In contrast to Example 12, the right state is only explored once, and an irrelevant intermediate state is eliminated.

For simplicity, we have decided to showcase the time-aware train selection strategy with the single-edge path extension. However, it can naturally also be applied to any other extension strategy presented in Section 4.2.

³ The previous approach “selects” all trains whose path has not yet been fully specified.

⁴ In our implementation, we decide to expand trains with higher importance first.



■ **Figure 5** Time-Aware Train Selection.

By ordering the trains by time as proposed in Section 4.1, we ensure that no state is reexplored and the search space becomes a tree. However, the attentive reader might realize that we also cut off valid goal states by doing so. In the previous approach, the train order on a section was determined by whose path was extended onto that section first. Without an additional ordering rule, the order would now be determined by timing, with tracks assigned on a first-come, first-served basis. At the same time, a train may need to wait to let an opposing train clear a section or to let a faster train pass by. Without further refinement, such solutions are pruned and cannot be reached by the search algorithm.

Since it is no longer possible to freely decide which train to advance next, we need to make the ordering decision at a later time. Instead of appending a train to the end of the ordering, we branch into all possible orderings as soon as a new train is expanded into a TTD section. In contrast to the original non-time-aware case, it is now possible to schedule trains ahead of trains already assigned to a track segment.

► **Example 14.** Consider Figure 6a with the same situation as in Example 10. Instead of branching at the left state, tr_2 is expanded onto the shared section first because, in the initial simulation, it arrives there earlier. After that, tr_1 is expanded onto the shared section. To not lose any valid solution, we branch at this point, allowing tr_1 to either go first or second.

We can already exclude some of the orderings, though, which we do not have to expand to. Note that for two trains following each other, a decision on their order is made the first time they enter a shared track segment. As long as their paths do not diverge after that, their order cannot change. Hence, we can forward this choice to later TTD sections without the need for further branching.

► **Example 15.** Consider Figure 6b and assume that tr_3 is expanded. On the right section $TTD2$, it is already decided that tr_1 travels before tr_2 . In theory, tr_3 could go first, second, or third. However, since it shares the path with tr_2 , their relative ordering compared to $TTD1$ cannot change. Hence, tr_3 cannot go last, and this variant is not generated as a successor state.

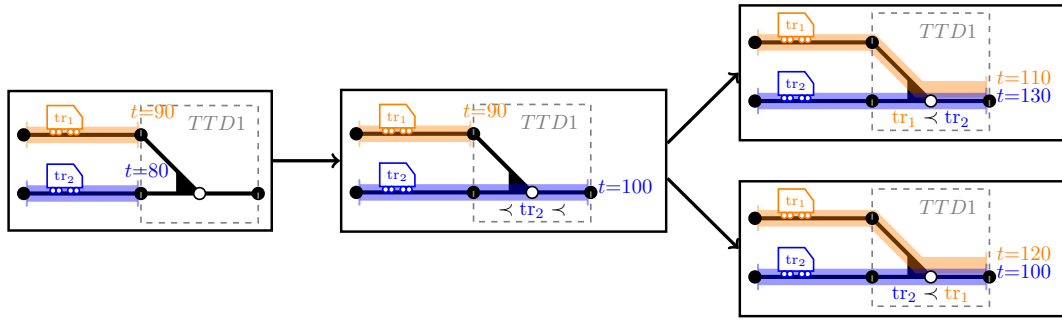
In order to still be able to reach all goal states, we need to slightly modify the simulation assumptions from Section 3.1.1 in two ways:

1. A train that disappeared does *not* block its last vertex.
2. Trains moving in opposite directions on a single track do *not* collide.

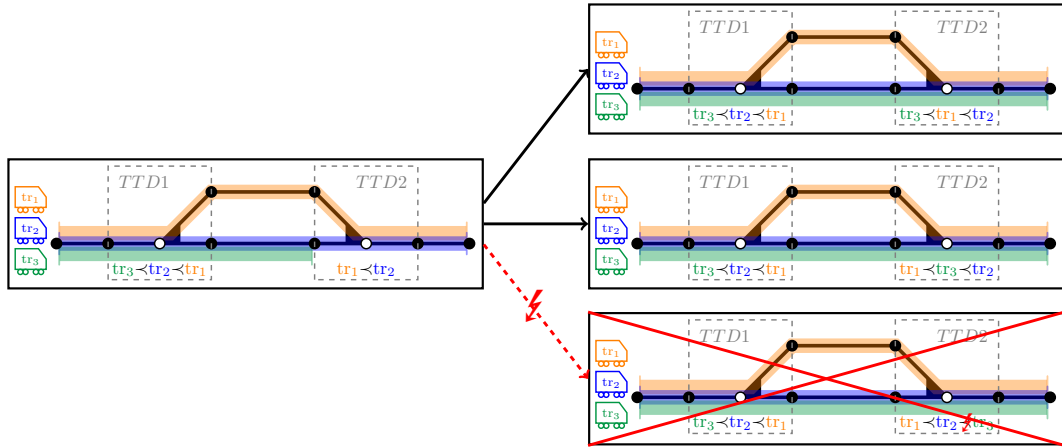
While these assumptions seem counterintuitive, not relaxing the simulation in these points would cut off feasible solutions (see proof of Theorem 17). It is worth noting that these changes do not affect the outcome of goal states, because

1. there are no trains with partial routes that disappear within the network, and
2. the train ordering on the TTD sections adjacent to a single track already prevents head-on collisions.

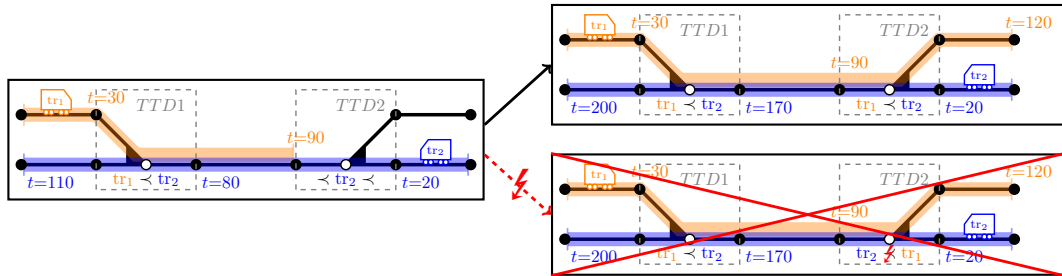
In particular, the relaxed simulation affects only intermediate states that are not returned as solutions; the simulation behavior for returned solutions remains unchanged.



(a) State Transitions.



(b) Deduction of Possible Train Ordering (Same Direction).



(c) Deduction of Possible Train Ordering (Opposite Direction).

■ **Figure 6** Time-Aware Train Ordering.

► Example 16.

1. Consider the situation from Example 15, where (without the first relaxation) the left state in Figure 6b would be infeasible, because tr_3 (as going first) would prevent tr_2 from reaching its destination. However, the state might be needed (depending on the train's timing) to reach two valid goal states.
2. Consider Figure 6c. Without the second relaxation, the left state would likely be infeasible due to a head-on collision on the shared track. Again, it might be needed to reach the valid goal state on the right, though. Note that due to the ordering deduction exemplified

in Example 15, no (invalid) successor state with ordering $\text{tr}_2 \prec \text{tr}_1$ on *TTD1* is created. Because of this, the TTD ordering prevents head-on collisions in goal states without further consideration.

It remains to show that the defined new state space is correct (i.e., all goal states can be reached) and prevents the undesired state reexplorations discussed in Section 3.3.

► **Theorem 17.** *In the (time-aware) state space as defined in this section, all feasible goal states can be reached from the initial state.*

Proof. Let $s_g \in \mathcal{G}$ be any feasible goal state and let $B_1^{(i)}, \dots, B_{k_i}^{(i)}$ be the TTD sections traversed by train $i \in \mathcal{T}$ in that goal state. Further let $B_0^{(i)} := u_{in}^{(i)} \in V$ be the train's entry vertex and $B_{k_i+1}^{(i)} := u_{out}^{(i)} \in V$ be the train's exit vertex.

Consider any prefix-state s of s_g , i.e., a partial solution (see Definition 7) where every train's subpath can be extended to the full path in s_g . The orders on all TTD sections and boundary vertices are inherited from s_g by restriction to trains whose partial routes reach the respective element.

Assume that s is not feasible, i.e., the simulation of s reaches a deadlock with at least one blocked train. For a blocked train i , choose one active obstruction $\eta_i = \text{clear}(p_i, j)$, i.e., i is currently blocked because another train j has not yet cleared a point, section, or vertex p_i . Since s_g is feasible, all η_i have been cleared in the goal state, denote the time of clearing in s_g by $t_g(\eta_i)$.

Choose a blocked train i^* with minimal $t_g(\eta_{i^*})$, where $\eta_{i^*} = \text{clear}(p_{i^*}, j)$, i.e., i^* is blocked by train j which has not cleared p_{i^*} during the simulation of state s , hence, j cannot advance. Because trains are assumed to disappear at the end of their partial route without blocking their last vertex indefinitely, j itself must be blocked by some train $k \neq j^5$ which has not yet cleared some p_j . Thus $\eta_j = \text{clear}(p_j, k)$.

We claim that the relative order of the conflicting trains at p_{i^*} and p_j is identical in s and s_g . There are two cases:

1. A train can be blocked by another train because it has not cleared a given TTD section or boundary vertex. In this case, the train order is inherited explicitly by definition.
2. A train can be blocked by another train traveling in the same direction on any intermediate edge. Since trains cannot overtake each other on the same directed track segment, the train order is uniquely determined by the order at the previous TTD section or boundary vertex (which both trains have in common and already passed even in state s)⁶.

Since s_g is feasible and the relative order is identical, j eventually clears p_{i^*} in s_g . However, because j requires p_j to be clear to advance out of p_{i^*} , p_j had to be cleared by k strictly before that. Hence,

$$t_g(\eta_j) = t_g(\text{clear}(p_j, k)) < t_g(\text{clear}(p_{i^*}, j)) = t_g(\eta_{i^*}), \quad (5)$$

which is a contradiction to $t_g(\eta_{i^*})$ being minimal.

In particular, all prefix-states of s_g are valid under the modified simulation assumption for partial states.

⁵ $k = i^*$ is possible, but the argument is independent from whether $k = i^*$ or $k \neq i^*$.

⁶ This argument would not work for trains traveling in opposite directions, because they do not share a common previous TTD section. If their partial routes end between two sections, their order might not be uniquely deducible for state s . Hence, the second relaxation to the simulator described above is crucial for the correctness of the A* search.

It remains to show that there is a path from the initial state to s_g in the time-aware state space that uses only prefix-states. Start with the initial state, which is a prefix-state of s_g . Now consider any reachable prefix-state $s \neq s_g$. The time-aware train selection rule (Section 4.1) selects exactly one train that has not yet been fully specified. Apply the path extension that follows the route and stop choices of this train in s_g and, whenever a newly entered TTD section creates multiple ordering branches, choose the branch matching the ordering of s_g . The resulting successor s^+ is again a prefix-state of s_g . Each such step strictly increases the amount of specified information (at least one additional edge and, if applicable, additional stop information), and there are only finitely many edges and scheduled stops. Hence, s_g is finally reached, and there is a path from the initial state to s_g in the time-aware state space. ◀

► **Theorem 18.** *The (time-aware) state space as defined in this section is a (directed) tree.*

Proof. Assume the contrary, then there is a state s with two distinct successors s_1^+ and s_2^+ and a state $\hat{s}$ such that in the state space there is a path from both s_1^+ and s_2^+ to $\hat{s}$. Since s is only expanded on a single train⁷, say tr_1 , in the time-aware setting, s_1^+ and s_2^+ can only differ by the routing data (i.e., path, stop position, or relative train ordering on some TTD section or boundary vertex) of tr_1 . However, this difference would not be reverted (since the routing data is only extended and not changed in future states) and, hence, be forwarded up to $\hat{s}$, which is a contradiction. ◀

4.2 Path Extensions

[4] proposes two ways to extend paths of a given train. The basic idea is to move trains one edge at a time. At a turnout, this leads to multiple successor states; besides that, the path extension is unique.

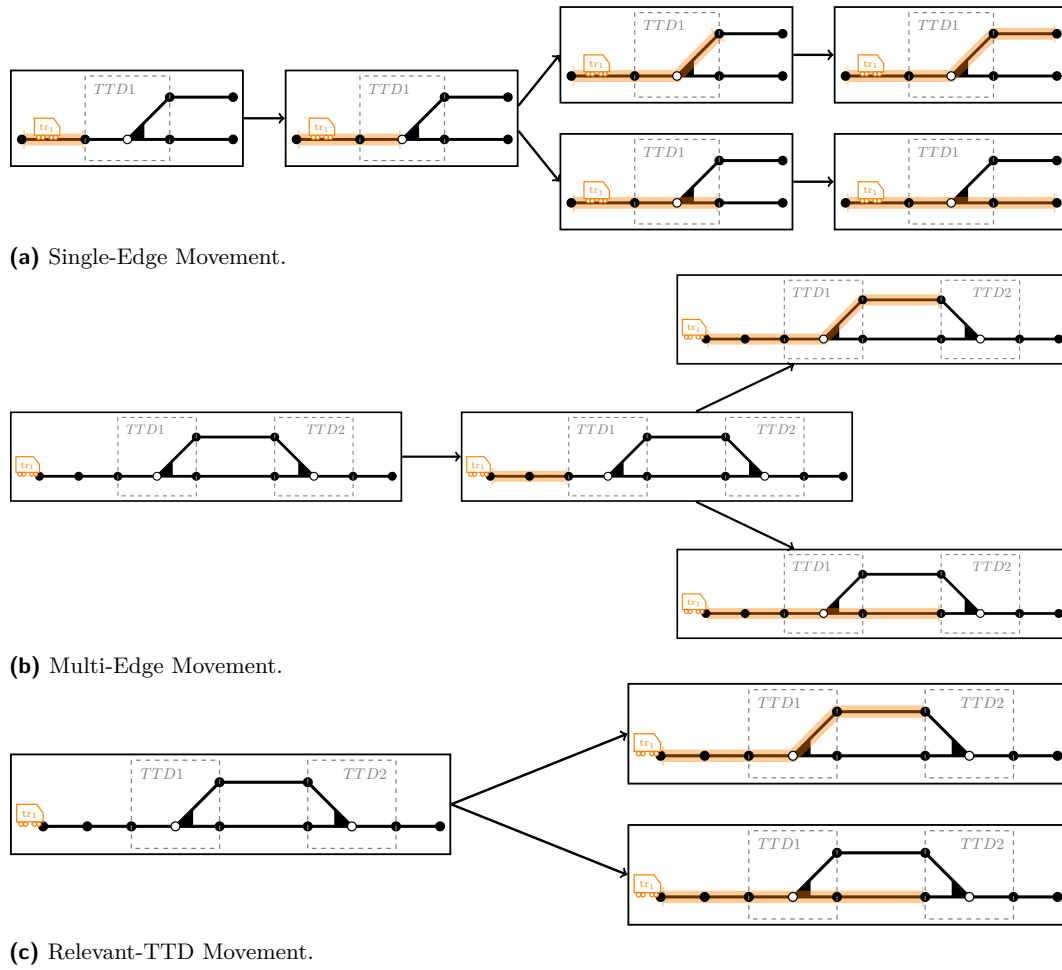
► **Example 19.** Consider Figure 7a with a single train. Neighboring states differ by exactly one edge. At the turnout's vertex, two possible successors exist: the train can either turn left or stay on the main line.

Often, the path extension is unique. Hence, it seems sensible to skip some intermediate steps that do not add value. Recall that in [4] the train ordering is deduced by whichever train extends on an edge first. Thus, we can safely combine multiple edges into a single transition as long as the train ordering is the same across all added edges. On a single track, trains cannot overtake each other. Because of this, [4] proposes to extend train routes up to the next turnout's TTD section.

► **Example 20.** Consider Figure 7b with a single train that has not entered the network yet. First, the train can enter up to TTD1, i.e., the first turnout. It can then transition to either move left or stay on the main line. However, the choice is directly extended up to the second turnout (TTD2).

Looking at Example 20, it is apparent that stopping at TTD1 is not necessary, since any other train trying to cut in must use the edge already blocked by the considered train. If the network consisted of additional trains, these might be moved in intermediate states before the considered train advances. Hence, we propose to further reduce the state space. A train's path extension shall only end at "relevant" TTD sections, i.e., a TTD section that can be traversed by another train without using the entry edge blocked by the current train.

⁷ This did not hold for the state space introduced by [4], which caused the reexploration issue described in Section 3.3.



■ **Figure 7** State Transitions.

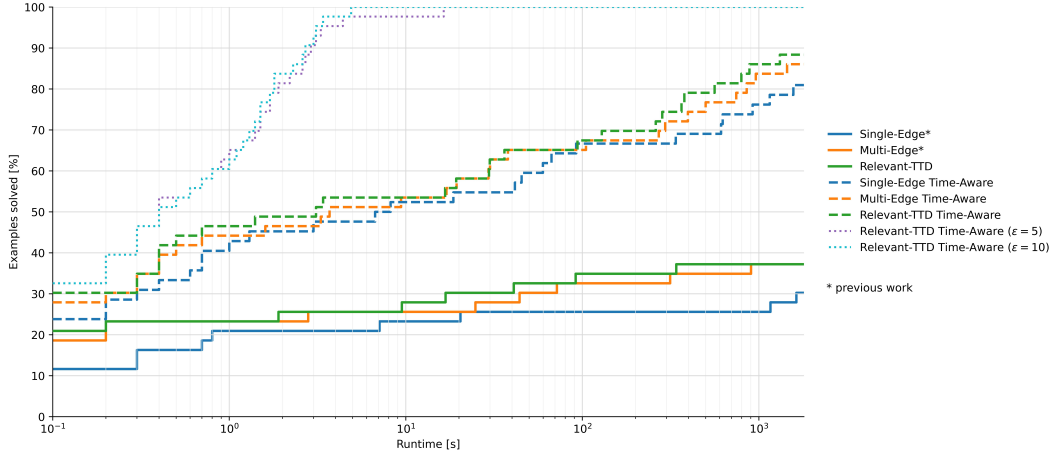
► **Example 21.** Consider Figure 7c with the same initial state as in Example 20. Instead of adding an intermediate state, where the train moves to the first turnout, two successor states are generated directly, both moving to the first “relevant” turnout.

5 Case Study

We extend the implementation from [4] by the adaptations described in Section 4. The proposed method is implemented in C++ and is open-source as part of the *Munich Train Control Toolkit* (MTCT), available on GitHub at <https://github.com/cda-tum/mtct>.

For testing, we use the benchmark from [4], which is also available in the above-mentioned repository. It contains several network families, including instances inspired by the Munich Stammstrecke; these instances have comparatively few routing alternatives but an increasing number of trains. Different variants of the algorithm were tested on an Intel(R) Xeon(R) W-1370P system with a 3.60GHz CPU (8 cores) and 128GB of RAM, running Ubuntu 20.04.

As discussed, the search variants differ in two aspects. First, they differ in the rule used to select the next train whose path is expanded. For this, we compare the strategy from [4] with the time-aware train selection strategy proposed in Section 4.1. In both cases, the relative train orderings are determined by that train selection rule.



■ **Figure 8** Runtime comparison.

For each of the two train selection strategies, we test three path extension strategies: the two already proposed by [4] and the extension discussed in Section 4.2.

- *Single-Edge*: A train’s path is extended by exactly one edge at a time, see Example 19. This strategy was originally proposed by [4].
- *Multi-Edge*: A train’s path is extended by multiple edges at a time until arriving at the next TTD section (usually a turnout), see also Example 20. This strategy was originally proposed by [4].
- *Relevant-TTD*: Similarly to *Multi-Edge*, however, a path extension does not end at a section that cannot be traversed without using the edge on which the train in question approaches it (hence, the train cannot be required to wait there for a train on a different path), see also Example 21.

We ran all instances with a 30-minute (1800-second) timeout. The results are shown in Figure 8. The x-axis denotes runtime in seconds and is log-based, so both easy and hard instances can be compared in a single plot. The y-axis shows the percentage of instances that could be solved within the given time. Hence, in general, lines to the left/top correspond to faster algorithms. The raw data can be found in Appendix A and contains information on runtimes, gaps, the number of (distinct) states simulated, and the number of states that were reexplored.

We were able to reproduce the results reported in [4]. Consider the solid lines in Figure 8 that represent the search algorithm without the time-aware train selection proposed in this work. Again, extending states by more than one edge at a time improved the runtime notably. At the same time, the improvements do not allow solving significantly more instances. Most notably, all variants described in [4] struggle to route trains on instances inspired by the Munich Stammstrecke, even though these instances offer only a limited set of routing choices. This indicates that the previous approach scales badly with the number of trains. In Appendix A, we provide information on how many states were reexplored. In the extreme case, approximately 58% of explored states had been seen and simulated before. This confirms that the problem described in Section 3.3 is not only theoretical, but of practical relevance.

A significantly larger runtime improvement comes from the time-aware train selection strategy (dashed lines) proposed in Section 4.1. On the Munich Stammstrecke instances mentioned above, the resulting runtimes are consistent with the limited number of routing

alternatives. An instance with 16 trains, which timed out in [4], can now be solved in approximately 1 second. While this is an extreme case, it illustrates how strongly reexploring states can affect performance. The same effect, though less pronounced, is also evident across all other tested networks.

All variants discussed so far retain the same *theoretical* optimality guarantee under Assumption 11 [4, Corollary 17]. Consistent with that theory, changing the strategy affected the runtime but did not influence the objective value of the returned solution on the tested instances.

Finally, we ran the best-performing exact strategy with higher weights $\varepsilon \in \{5, 10\}$. The search algorithm is guided more towards goal states, and as a result, the result is no longer guaranteed to be optimal. With this, we were able to find feasible solutions to all tested instances within 20 seconds. Wherever the true optimum was known, we compared the objective values. In the worst case, the objective value doubled. On the other hand, the median increase in objective value was around 9.5% ($\varepsilon = 5$) and 14% ($\varepsilon = 10$) respectively.

6 Conclusions

In this work, we have extended an algorithm for optimal train routing based on A^* [4] by significantly reducing the state space. We conclude that the previous version frequently expanded to states it had already explored before (Section 3.3). Because of this, an alternative state space was proposed by introducing a time-aware train selection rule (Section 4.1). The resulting state space is a (directed) tree, preventing undesired reexploration.

Section 5 showed that the addressed issue had a significant impact on runtimes. Across a small benchmark set, solving times are significantly reduced while still returning solutions with the same objective values as the corresponding exact variants and while retaining the same conditional optimality guarantee. Moreover, the approach remains flexible, and the simulator can (in theory) be replaced by any other simulation system satisfying Assumption 11, e.g., a more efficient event-based simulator. Additionally, this allows easy adaptation to other and future control systems. We provide all implementations open-source as part of the *Munich Train Control Toolkit*, available on GitHub at <https://github.com/cda-tum/mtct>.

At the same time, we still experienced significant runtimes in some instances. Approximate search algorithms on the proposed state space could provide further improvements, but at the cost of losing optimality guarantees. A natural choice is to use a weighted version of A^* that gives more weight to the heuristic and, hence, pushes the algorithm towards goal states more quickly. While this showed a clear additional improvement in efficiency, optimality is no longer guaranteed. A more detailed discussion on this trade-off is out of scope for this paper. Exploring suitable approximate search algorithms for the proposed state space is left for further research.

Overall, this work shows the importance of design choices when defining a state space used with search algorithms. One should take extra care to ensure that, in every iteration, the algorithm extends only towards states that provide new information the search has not revealed before. In our case (since the objective is entirely defined by the state itself, not the transitions), the search space should be a tree. Achieving this requires making counterintuitive design choices (Example 16), but our case study shows that this (likely) pays off significantly.

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. doi:10.1007/978-3-319-72153-8.
- 2 Stefan Engels and Robert Wille. Munich Train Control Toolkit. Software, swhId: swh:1:dir:2ffb77c8877a842cbc78f45016d4270dd5aeff5d (visited on 2026-09-22). URL: <https://github.com/cda-tum/mtct>, doi:10.4230/artifacts.28078.
- 3 Stefan Engels and Robert Wille. Comparing lazy constraint selection strategies in train routing with moving block control. In *Proceedings of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS)*, volume 39, pages 585–590. Polish Information Processing Society, 2024. doi:10.15439/2024f3041.
- 4 Stefan Engels and Robert Wille. Using A* for optimal train routing on moving block systems. In *25th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2025, Warsaw, Poland, September 18-19, 2025*, volume 137, pages 14:1–14:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/OASICS.ATMOS.2025.14.
- 5 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. doi:10.1109/TSSC.1968.300136.
- 6 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. doi:10.1109/nano61778.2024.10628808.
- 7 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 2022)*, volume 106. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/OASICS.ATMOS.2022.11.
- 8 Jörn Pachl. *Railway Signalling Principles: Edition 2.0*. Universitätsbibliothek Braunschweig, 2021. doi:10.24355/dbbs.084-202110181429-0.
- 9 Tom Peham, Judith Przigoda, Nils Przigoda, and Robert Wille. Optimal railway routing using virtual subsections. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*, pages 63–79. Springer International Publishing, 2022. doi:10.1007/978-3-031-05814-1_5.
- 10 Ira Pohl. First results on the effect of error in heuristic search. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 5*, pages 219–236. Edinburgh University Press, Edinburgh, 1970.
- 11 Thomas Schlechte, Ralf Borndörfer, Jonas Denißen, 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. doi:10.1016/j.jrtpm.2022.100315.
- 12 Lars Schnieder. *Communications-Based Train Control (CBTC)*. Springer Berlin Heidelberg, March 2021. doi:10.1007/978-3-662-62876-8.

A Raw Data

Instance	TA	ε	Single-Edge				Multi-Edge				Relevant-TTD						
			objective	gap [%]	time [s]	simulated	reexplored	objective	gap [%]	time [s]	simulated	reexplored	objective	gap [%]	time [s]	simulated	reexplored
High Speed Track (02 Trains)	no	1	648	0.00	< 0.01	5	0	648	0.00	< 0.01	5	0	648	0.00	< 0.01	5	0
	yes	5	648	0.00	< 0.01	3	0	648	0.00	< 0.01	3	0	648	0.00	< 0.01	3	0
	yes	10	-	-	-	-	-	-	-	-	-	-	648 (+0%)	0.00	< 0.01	3	0
High Speed Track (05 Trains)	no	1	792	0.00	0.01	81	0	792	0.00	0.01	81	0	792	0.00	0.01	81	0
	yes	1	792	0.00	< 0.01	6	0	792	0.00	< 0.01	6	0	792 (+0%)	0.00	< 0.01	6	0
	yes	5	-	-	-	-	-	-	-	-	-	-	792 (+0%)	0.00	< 0.01	6	0
Overtake	no	1	458	0.00	0.76	7958	5781	458	0.00	0.09	952	421	458	0.00	0.08	1010	171
	yes	1	458	0.00	0.01	119	0	458	0.00	< 0.01	53	1	500 (+9%)	9.00	< 0.01	19	2
	yes	5	-	-	-	-	-	-	-	-	-	-	600 (+31%)	24.2	< 0.01	17	2
Simple 2-Track Station	no	1	526	0.00	0.20	3357	1737	526	0.00	0.01	268	49	526	0.00	0.01	255	18
	yes	1	526	0.00	< 0.01	125	0	526	0.00	< 0.01	79	0	526	0.00	< 0.01	73	0
	yes	5	-	-	-	-	-	-	-	-	-	-	526 (+0%)	5.89	< 0.01	33	0
Simple Network	no	1	975	0.00	7.06	33206	24128	975	0.00	0.19	880	352	975	0.00	0.05	105	41
	yes	1	975	0.00	0.17	530	0	975	0.00	0.11	328	0	975	0.00	0.03	134	0
	yes	10	-	-	-	-	-	-	-	-	-	-	1873 (+92%)	47.9	< 0.01	33	0
Simple Network (03 Random Trains)	no	1	900	0.00	0.21	1196	637	900	0.00	0.03	138	40	900	0.00	0.01	63	7
	yes	1	900	0.00	0.01	53	0	900	0.00	< 0.01	33	0	900	0.00	< 0.01	37	0
	yes	10	-	-	-	-	-	-	-	-	-	-	1818 (+102%)	50.5	< 0.01	20	0
Simple Network (06 Random Trains)	no	1	1778	0.00	1632	3905378	2216337	1778	0.00	24.8	64087	19052	1778	0.00	9.45	23712	4009
	yes	1	1778	0.00	8.16	11309	0	1778	0.00	3.67	5270	0	1778 (+0%)	12.2	0.18	312	0
	yes	5	-	-	-	-	-	-	-	-	-	-	2971 (+67%)	47.4	0.12	163	0
Simple Network (09 Random Trains)	no	1	-	-	timeout	6803640	1721194	2131	0.00	903	1919196	371207	2131	0.00	341	736508	71154
	yes	1	2131	0.00	18.6	17104	0	2131	0.00	9.31	8649	0	2131	0.00	3.31	3343	0
	yes	5	-	-	-	-	-	-	-	-	-	-	2132 (+0%)	5.63	0.29	306	0
Simple Network (12 Random Trains)	no	1	-	-	timeout	7144204	1353229	-	-	timeout	3583778	644052	-	-	timeout	2558439	219886
	yes	1	3485	0.00	1569	900753	0	3485	0.00	745	467117	0	2974	0.00	129	87299	0
	yes	5	-	-	-	-	-	-	-	-	-	-	2988 (+0%)	26.1	1.68	1536	0
Simple Network (15 Random Trains)	no	1	-	-	timeout	6454866	990426	-	-	timeout	5216174	730078	-	-	timeout	3991428	230888
	yes	1	-	-	timeout	847619	0	-	-	timeout	828466	0	3004	0.00	378	290355	0
	yes	5	-	-	-	-	-	-	-	-	-	-	3223 (+7%)	17.0	16.3	4676	0
Simple Network (18 Random Trains)	no	1	-	-	timeout	6046797	581883	-	-	timeout	534558	534558	-	-	timeout	4380981	299054
	yes	1	-	-	timeout	912273	0	-	-	timeout	860678	0	-	-	timeout	872613	0
	yes	10	-	-	-	-	-	-	-	-	-	-	3818	20.9	1.14	474	0
Single Track With Station	no	1	470	0.00	< 0.01	74	4	470	0.00	< 0.01	21	0	470	0.00	< 0.01	21	0
	yes	1	470	0.00	< 0.01	12	0	470	0.00	< 0.01	6	0	470	0.00	< 0.01	6	0
	yes	5	-	-	-	-	-	-	-	-	-	-	470 (+0%)	0.00	< 0.01	6	0
Single Track Without Station	no	1	648	0.00	< 0.01	5	0	648	0.00	< 0.01	5	0	648	0.00	< 0.01	5	0
	yes	1	648	0.00	< 0.01	3	0	648	0.00	< 0.01	3	0	648	0.00	< 0.01	3	0
	yes	5	-	-	-	-	-	-	-	-	-	-	648 (+0%)	0.00	< 0.01	3	0
Stammstrecke (01 Random Trains)	no	1	690	0.00	< 0.01	37	0	690	0.00	< 0.01	13	0	690	0.00	< 0.01	10	1
	yes	1	690	0.00	< 0.01	37	0	690	0.00	< 0.01	13	0	690	0.00	< 0.01	10	1
	yes	5	-	-	-	-	-	-	-	-	-	-	690 (+0%)	0.00	< 0.01	10	1
Stammstrecke (02 Random Trains)	no	1	1047	0.00	0.02	2699	1604	1047	0.00	0.15	692	275	1047	0.00	0.13	567	270
	yes	1	1047	0.00	0.02	106	0	1047	0.00	0.01	56	0	1047	0.00	0.01	54	8
	yes	5	-	-	-	-	-	-	-	-	-	-	1047 (+0%)	0.00	0.01	54	8

Instance	TA	ε	Single-Edge			Multi-Edge			Relevant-TTD			
			objective	gap [%]	time [s]	simulated	recexplored	objective	gap [%]	time [s]	simulated	recexplored
Stammstrecke (03 Random Trains)	no	1	999	0.00	20.3	66432	63526	999	0.00	2.73	8990	5173
	yes	1	999	0.00	0.06	147	-	999	0.00	0.02	80	0
	yes	5	-	-	-	-	-	999 (+0%)	0.00	-	-	-
Stammstrecke (04 Random Trains)	yes	10	-	-	-	-	-	999 (+0%)	0.00	0.03	75	9
	no	1	-	-	timeout	4157345	5658774	1077	0.00	44.1	100167	72883
	yes	1	1077	0.00	0.25	469	0	1077 (+6%)	0.00	0.10	198	0
Stammstrecke (04 Trains)	yes	5	-	-	-	-	-	1077 (+6%)	6.20	0.04	77	11
	yes	10	-	-	-	-	-	1077 (+6%)	6.20	0.04	77	11
Stammstrecke (05 Random Trains)	no	1	1142	0.00	1164	2413320	2843988	1142	0.00	71.8	150435	104013
	yes	1	1142	0.00	0.11	269	0	1142	0.00	0.06	112	0
	yes	5	-	-	-	-	-	1284 (+12%)	11.2	0.07	111	21
Stammstrecke (06 Random Trains)	yes	10	-	-	-	-	-	1284 (+12%)	11.2	0.07	111	21
	no	1	-	-	timeout	5664386	6619922	1004	0.00	314	709028	424182
	yes	1	1004	0.00	0.39	806	0	1004	0.00	0.21	389	0
Stammstrecke (06 Trains)	yes	5	-	-	-	-	-	1145 (+14%)	13.9	0.07	124	31
	yes	10	-	-	-	-	-	1145 (+14%)	15.9	0.07	124	31
Stammstrecke (07 Random Trains)	no	1	-	-	timeout	4999473	3930995	-	-	timeout	3529632	235153
	yes	1	1208	0.00	0.94	1387	0	1208	0.00	0.47	688	0
	yes	5	-	-	-	-	-	1360 (+13%)	15.7	0.13	164	32
Stammstrecke (07 Trains)	yes	10	-	-	-	-	-	1360 (+13%)	15.7	0.12	164	32
	no	1	-	-	timeout	5691200	3306782	-	-	timeout	4102183	2484273
	yes	1	1125	0.00	0.66	956	0	1125	0.00	0.35	464	0
Stammstrecke (08 Random Trains)	yes	5	-	-	-	-	-	1253 (+11%)	13.4	0.13	164	26
	yes	10	-	-	-	-	-	1253 (+11%)	13.4	0.13	164	26
	no	1	-	-	timeout	6432961	3608147	-	-	timeout	3680650	1841907
Stammstrecke (08 Trains)	yes	1	1118	0.00	0.67	855	0	1118	0.00	0.34	403	0
	yes	5	-	-	-	-	-	1278 (+14%)	14.1	0.21	202	29
	yes	10	-	-	-	-	-	1278 (+14%)	14.1	0.22	202	29
Stammstrecke (08 Trains)	no	1	-	-	timeout	2525016	153747	-	-	timeout	2788664	1523112
	yes	1	1187	0.00	0.51	420	0	1187	0.00	0.27	228	0
	yes	5	-	-	-	-	-	1436 (+21%)	17.5	0.30	230	43
Stammstrecke (09 Random Trains)	yes	10	-	-	-	-	-	1436 (+21%)	17.5	0.29	230	43
	no	1	-	-	timeout	7315637	3245949	-	-	timeout	4612115	2085263
	yes	1	1098	0.00	1.24	1405	0	1098	0.00	0.61	688	0
Stammstrecke (10 Random Trains)	yes	5	-	-	-	-	-	1179 (+7%)	10.5	0.23	221	51
	yes	10	-	-	-	-	-	1301 (+18%)	18.8	0.25	225	53
	no	1	-	-	timeout	6824701	2484781	-	-	timeout	4788169	1965229
Stammstrecke (10 Trains)	yes	1	1264	0.00	6.64	5288	0	1264	0.00	3.21	2417	0
	yes	5	-	-	-	-	-	1407 (+16%)	19.1	0.37	243	46
	yes	10	-	-	-	-	-	1452 (+15%)	18.2	0.36	247	49
Stammstrecke (11 Random Trains)	no	1	-	-	timeout	6363719	2474871	-	-	timeout	4739460	1660505
	yes	1	1240	0.00	66.9	54523	0	1240	0.00	29.4	23504	0
	yes	5	-	-	-	-	-	1412 (+14%)	19.0	0.38	278	58
Stammstrecke (11 Trains)	yes	10	-	-	-	-	-	1345 (+8%)	15.0	0.37	278	58
	no	1	-	-	timeout	6881161	1516836	-	-	timeout	3386717	843681
	yes	1	1253	0.00	41.5	27533	0	1253	0.00	19.3	12328	0
Stammstrecke (12 Random Trains)	yes	5	-	-	-	-	-	1484 (+18%)	18.8	0.53	329	71
	yes	10	-	-	-	-	-	1484 (+18%)	18.8	0.53	329	71
	no	1	-	-	timeout	6933154	2452527	-	-	timeout	4855449	1191177
Stammstrecke (13 Random Trains)	yes	1	1329	0.00	59.8	36411	0	1329	0.00	29.9	17775	0
	yes	5	-	-	-	-	-	1452 (+9%)	12.0	0.72	368	81
	yes	10	-	-	-	-	-	1452 (+9%)	12.0	0.72	368	81
Stammstrecke (14 Random Trains)	no	1	-	-	timeout	4191645	1088269	-	-	timeout	3675403	940647
	yes	1	1321	0.00	92.2	40272	0	1321	0.00	37.9	20220	0
	yes	5	-	-	-	-	-	1416 (+7%)	10.7	0.68	349	58
	yes	10	-	-	-	-	-	1416 (+7%)	10.7	0.67	349	58

Instance	TA	ε	Single-Edge				Multi-Edge				Relevant-TTD						
			objective	gap [%]	time [s]	simulated	recexplored	objective	gap [%]	time [s]	simulated	recexplored	objective	gap [%]	time [s]	simulated	recexplored
Stammstrecke (15 Random Trains)	no	1	-	-	timeout	7077858	1337835	-	-	timeout	3238225	953080	-	-	timeout	3096845	1402920
	yes	1	1381	0.00	45.2	23246	0	1381	0.00	16.9	8562	0	1381	0.00	16.6	8015	1656
	yes	5	-	-	-	-	-	-	-	-	-	-	1513 (+10%)	10.5	0.81	335	62
	yes	10	-	-	-	-	-	-	-	-	-	-	1851 (+34%)	26.8	0.91	335	62
Stammstrecke (16 Random Trains)	no	1	-	-	timeout	7039892	1575351	-	-	timeout	3850785	666673	-	-	timeout	3878295	803242
	yes	1	1418	0.00	338	141626	0	1418	0.00	105	43343	0	1418	0.00	93.8	38655	4264
	yes	5	-	-	-	-	-	-	-	-	-	-	1689 (+19%)	19.6	0.98	354	52
	yes	10	-	-	-	-	-	-	-	-	-	-	1743 (+23%)	22.1	1.03	366	52
Stammstrecke (16 Trains)	no	1	-	-	timeout	2221055	644207	-	-	timeout	2141924	568399	-	-	timeout	2023531	738399
	yes	1	1383	0.00	2.95	949	0	1383	0.00	1.51	478	0	1383	0.00	1.35	488	97
	yes	5	-	-	-	-	-	-	-	-	-	-	1754 (+27%)	21.2	1.68	498	99
	yes	10	-	-	-	-	-	-	-	-	-	-	1763 (+28%)	21.6	1.74	486	93
Stammstrecke (17 Random Trains)	no	1	-	-	timeout	2963531	519844	-	-	timeout	3242234	656299	-	-	timeout	3127460	923632
	yes	1	-	-	-	-	-	1517	0.00	961	273975	-	1517	0.00	885	248421	32107
	yes	5	-	-	-	-	-	-	-	-	-	-	1745 (+15%)	18.7	1.46	425	57
	yes	10	-	-	-	-	-	-	-	-	-	-	1745 (+15%)	18.7	1.46	425	57
Stammstrecke (18 Random Trains)	no	1	-	-	timeout	5176165	933973	-	-	timeout	4620655	950424	-	-	timeout	3721597	1021678
	yes	1	-	-	timeout	745673	0	1563	0.00	1445	537518	0	1563	0.00	1316	475233	58127
	yes	5	-	-	-	-	-	-	-	-	-	-	1850 (+18%)	19.4	1.46	460	69
	yes	10	-	-	-	-	-	-	-	-	-	-	1974 (+26%)	24.4	1.44	456	69
Stammstrecke (19 Random Trains)	no	1	-	-	timeout	3948088	891329	-	-	timeout	2983874	621804	-	-	timeout	2910194	745411
	yes	1	1482	0.00	922	313367	0	1482	0.00	396	132275	0	1482	0.00	365	117502	16305
	yes	5	-	-	-	-	-	-	-	-	-	-	1674 (+13%)	14.0	1.39	463	73
	yes	10	-	-	-	-	-	-	-	-	-	-	1629 (+10%)	11.6	1.37	463	73
Stammstrecke (20 Random Trains)	no	1	-	-	timeout	5572922	1203984	-	-	timeout	3700443	532509	-	-	timeout	3436993	1050945
	yes	1	1452	0.00	611	213008	0	1452	0.00	271	92605	0	1452	0.00	261	88205	12584
	yes	5	-	-	-	-	-	-	-	-	-	-	1820 (+25%)	22.5	1.81	484	64
	yes	10	-	-	-	-	-	-	-	-	-	-	1785 (+23%)	20.9	1.76	478	64
Stammstrecke (21 Random Trains)	no	1	-	-	timeout	3377181	703534	-	-	timeout	3689337	539420	-	-	timeout	4214797	876181
	yes	1	-	-	timeout	520273	0	1624	0.00	854	220851	0	1624	0.00	796	216250	38871
	yes	5	-	-	-	-	-	-	-	-	-	-	1873 (+15%)	17.5	2.14	564	100
	yes	10	-	-	-	-	-	-	-	-	-	-	1959 (+21%)	21.2	2.25	568	102
Stammstrecke (22 Random Trains)	no	1	-	-	timeout	4434122	700002	-	-	timeout	3821167	637203	-	-	timeout	2935189	799919
	yes	1	1540	0.00	624	157447	0	1540	0.00	295	73885	0	1540	0.00	284	71149	13204
	yes	5	-	-	-	-	-	-	-	-	-	-	1703 (+11%)	12.4	2.53	608	97
	yes	10	-	-	-	-	-	-	-	-	-	-	1740 (+13%)	14.3	2.51	606	95
Stammstrecke (23 Random Trains)	no	1	-	-	timeout	5328391	577265	-	-	timeout	4583984	549486	-	-	timeout	4612401	897955
	yes	1	-	-	timeout	553884	0	-	-	timeout	476933	0	-	-	timeout	494822	99086
	yes	5	-	-	-	-	-	-	-	-	-	-	1764	10.3	2.85	684	146
	yes	10	-	-	-	-	-	-	-	-	-	-	1815	12.9	2.95	684	146
Stammstrecke (24 Random Trains)	no	1	-	-	timeout	2489865	406614	-	-	timeout	2413180	318846	-	-	timeout	2244219	460228
	yes	1	-	-	timeout	466052	0	-	-	timeout	382398	0	-	-	timeout	367264	71744
	yes	5	-	-	-	-	-	-	-	-	-	-	1835	13.3	2.62	618	113
	yes	10	-	-	-	-	-	-	-	-	-	-	1981	19.7	2.62	605	114
Stammstrecke (25 Random Trains)	no	1	-	-	timeout	4433915	582261	-	-	timeout	2862630	322555	-	-	timeout	3074197	523929
	yes	1	1519	0.00	1153	261561	0	1519	0.00	500	110481	0	1519	0.00	562	122062	23111
	yes	5	-	-	-	-	-	-	-	-	-	-	1653 (+49%)	10.9	3.01	654	106
	yes	10	-	-	-	-	-	-	-	-	-	-	1775 (+17%)	17.0	3.06	662	106
Stammstrecke (26 Random Trains)	no	1	-	-	timeout	3088864	301835	-	-	timeout	3906081	426125	-	-	timeout	2775562	517953
	yes	1	-	-	timeout	474812	0	-	-	timeout	407098	0	-	-	timeout	388653	43300
	yes	5	-	-	-	-	-	-	-	-	-	-	1935	15.3	3.28	598	66
	yes	10	-	-	-	-	-	-	-	-	-	-	1934	15.3	3.30	600	66
Stammstrecke (27 Random Trains)	no	1	-	-	timeout	4331628	462513	-	-	timeout	4019953	517792	-	-	timeout	3304114	568626
	yes	1	-	-	timeout	715059	0	-	-	timeout	583854	0	-	-	timeout	578196	136161
	yes	5	-	-	-	-	-	-	-	-	-	-	2008	14.7	4.40	796	177
	yes	10	-	-	-	-	-	-	-	-	-	-	2043	16.1	4.80	800	181