Optimal Railway Routing Using Virtual Subsections

Tom Peham[1], Judith Przigoda[2], Nils Przigoda[2], and Robert Wille[1,3]

[1] Chair for Design Automation, Technical University of Munich, Germany
[2] Siemens Mobility GmbH, Braunschweig, Germany
[3] Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, Austria
{tom.peham, robert.wille}@tum.de
{judith.przigoda, nils.przigoda}@siemens.com

Abstract. The design of railway systems has become a non-trivial task which more and more demands for efficient design automation methods. Modern railway systems based on standards such as the *European Train Control System* (ETCS) Level 3, the *Chinese Train Control System* (CTCS) Level 3+/4, or the Indian *Train Protection and Warning System* (TPWS) introduce new concepts such as virtual subsections which allow for a much higher degree of freedom and provide significant potential for increasing the efficiency in today's railway schedules. At the same time, this substantially increases the complexity of determining efficient solutions. The current state of the art addresses this complexity by discretizing the problem. In this work, we show that this, however, leads to substantial problems, namely infeasible configurations, rounding errors, and oversimplifications, that either harm the efficiency of the solving process or yield results which are significantly off from the actual optimum. Motivated by that, we propose an alternative design automation method that avoids discretization at all, overcomes the resulting problems, and additionally allows to solve the problem magnitudes faster than before.

1 Introduction

Railways are an important part of today's infrastructure, whether it is for delivering goods and resources or as a part of the public traffic system. They prove to be an environmentally friendly alternative to air traffic, road transport, and ship traffic. It is therefore vital to increase the usage of railways in the future – a goal which a huge number of societies has made one of their top priorities recently. But expanding railway infrastructures is costly, difficult, and time-consuming. The alternative is to increase the efficiency of existing railway infrastructure by increasing its throughput.

This can be achieved by putting more trains on the tracks. But to ensure the safe operation of them, railway networks are divided into *blocks*. A block can only be occupied by one train at any given time, thus, preventing collisions. To register trains moving in and out of blocks *Trackside Train Detection* (TTD) hardware, e. g., axle-counters, are employed. The blocks connected by TTD hardware are often also called TTDs and we will follow this convention in this paper.

A consequence of the resulting *block signaling* is that the throughput of a railway network is limited by the size of the blocks. Until a train has completely left a TTD, no other trains can enter the TTD. It is therefore sensible to decrease the size of TTDs in order to increase throughput. This entails installing new TTD hardware which requires maintenance and is not flexible when new layouts are required.

A solution to this problem is provided by the introduction of so-called *Virtual Subsections* (VSS). They are specified in modern railway traffic management systems such as the *European Train Control System* (ETCS) Level 3 [1, 2] by the *European Railway Traffic Management System* (ERTMS) [3,4], the *Chinese Train Control System* (CTCS) Level 3+/4 [5], or the Indian *Train Protection and Warning System* (TPWS) [6]. VSS essentially are blocks, just as TTDs. But in contrast to TTDs, these blocks do not require hardware. Instead, the occupation of a VSS is tracked by a radio control center which exchanges position information with trains in the network. Because these blocks are purely virtual, layouts are easy to adapt if changing schedules or demands necessitate it.

However, the implementation and utilization of such schemes and, hence, of virtual subsections, is just at the beginning. In fact, researchers started formalizing the underlying concepts (using, e. g., iUML-B [7,8], Electrum [9], SysML/KAOS [10], Event-B [11,12], or SPIN [13]), conducted corresponding case studies [14,15], or even presented first simulations [16,17]. But the main task, namely designing corresponding railway routings that exploit the extended degree of freedom provided by VSS in order to improve the travel times, remained an endeavor mostly tackled by manual labor thus far. Obviously, such a state of the art is not sufficient in order to address the upcoming challenges in extending the throughput of today's railway systems and, hence, *automatic* methods for railway routing using virtual subsections are urgently needed[1].

To the best of our knowledge, our previous approach recently introduced in [25] constitutes the first solution that generates (optimal) railway routings while, at the same time, using virtual subsections in order to minimize the sum of the travel times. To this end, we formulated the problem in terms of a satisfiability problem and, afterwards, used corresponding SAT solvers to determine a solution. This, however, requires a discretization of the problem which, on the one hand, makes the problem manageable for the solving engine, but also frequently leads to infeasible configurations, rounding errors, and oversimplifications. Hence, while providing a first solution towards design automation for

[1] Please note that, due to the long history of railway systems, approaches for routing trains through networks are of course not new and research into determining optimal schedules and verifying their correctness with respect to block signaling constraints has been conducted for a long time (see, e. g., [18–24]). Such solutions are inadequate when dealing with VSS because these approaches assume a fixed block layout to begin with. Simply partitioning the network into many VSS is also not practical because that puts a lot of workload on the radio control center communicating VSS occupations with the trains on the network. Solutions taking virtual subsections under consideration hardly exist.

modern railway routing, this approach still has severe shortcomings (something which is discussed and illustrated in more detail later in Section 3).

In this work, we propose an alternative solution for the railway routing problem using virtual subsections that overcomes these drawbacks. To this end, an A*-based search scheme is proposed which works *without* discretization but still is capable of efficiently determining optimal railway routings. Experiments confirm that the proposed scheme spares the user the need to determine a proper discrete formulation, generates results of much higher precision, and additionally is orders of magnitudes faster than the currently available solution.

The remainder of this paper is structured as follows: Section 2 briefly reviews the railway routing problem with VSS and Section 3 discusses the shortcomings of the currently available solution – motivating our work. Afterwards, the proposed solution is described in Section 4. Experimental evidence for the efficacy of the proposed solutions and comparisons to the state of the art is presented in Section 5. Section 6 concludes this paper.

2 Railway Routing in ETCS Level 3

This section briefly reviews and illustrates the considered problem as well as the used notation. We start with the concept of a *rail network* which can be modeled as an undirected edge-labeled graph $G = (V, E, L)$, where the edges E describe track sections that are connected via vertices V. The vertices $v \in V$ represent *Trackside Train Detection* (TTD) hardware like axle-counters (also called *TDD points*). The blocks separated by this hardware are often called TTDs themselves and we follow this convention in this work, i. e., we refer to edges E of G as TTDs[2]. Finally, every TTD $e \in E$ has a label $L(e) = l_e$ defining the length l_e of this TDD.

Example 1. Fig. 1 shows an example of a railway network with 4 TDD points and, hence, 7 TDDs labelled $e_1, e_2, \ldots, e_7$. The lengths l_e for all TDDs $e \in E$ are given by the labels annotated onto the respective blocks in Fig. 1.

To properly define train positions and train movements, we use the following terms: A *TTD interval* is a triple (e, a, b) with $e \in E$ and $0 \leq a < b \leq l_e$. A *range* is a sequence of TDD intervals $((e_1, a_1, b_1), \ldots, (e_n, a_n, b_n))$ such that, for all $1 \leq i < n$, e_i is connected to e_{i+1} via a TTD point in G and, for all $1 < i < n$, $a_i = 0$ and $b_i = l_{e_i}$. The length of a range rg is given by $l_{rg} = \sum_{i=1}^{n}(b_i - a_i) = (b_1 - a_1) + \sum_{i=2}^{n-1} l_{e_i} + (b_n - a_n)$. Then, a *train* tr is described by the tuple $tr = (l_{tr}, s_{tr}, start_{tr}, dest_{tr})$, where l_{tr} is the length of the train, s_{tr} its maximum speed, $start_{tr}$ its start position, and $dest_{tr}$ its destination position. A *position* p of a train tr is a range such that $l_p = l_{tr}$. Intuitively a train's position is the part of the track in the network that the train is standing on. The set of trains is denoted as Tr.

[2] We implicitly assumed here that the entire rail network being modelled is covered by TTDs. This is a reasonable assumption for modern rail network, however.

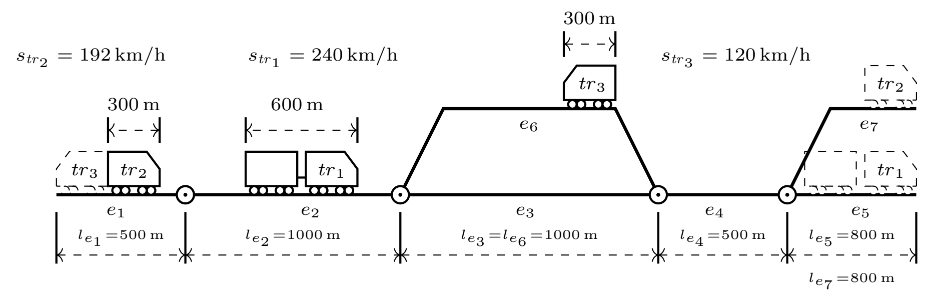


Fig. 1: Example network

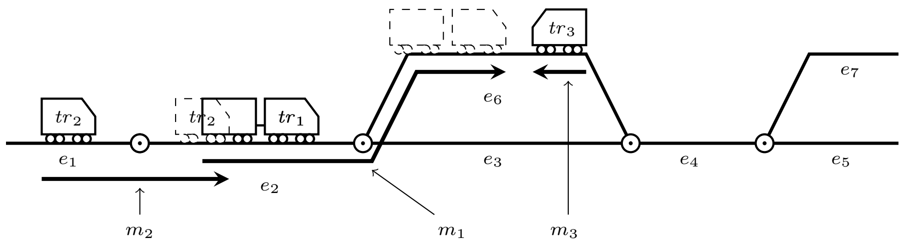


Fig. 2: Train movement

Example 2. The railway network shown in Fig. 1 is used by three trains: The train denoted tr_1 occupies a part of TTD e_2, i. e., its position is described by the range containing a single TTD interval $((e_2, 200\,\text{m}, 800\,\text{m}))$. The position of tr_2 is $((e_1, 100\,\text{m}, 400\,\text{m}))$. The position of tr_3 is $((e_6, 650\text{m}, 950\,\text{m}))$. Note that the length of the positions is the same as the lengths of the trains. The respective maximum speeds $s_{tr_1}, s_{tr_2}, s_{tr_3}$ are given above the trains in Fig. 1. The start positions are depicted by the trains with solid lines and the destination positions are depicted by trains with dashed lines. Train tr_1 has destination position $((e_5, 200\,\text{m}, 800\,\text{m}))$, tr_2 has destination position $((e_7, 500\,\text{m}, 800\,\text{m}))$, and tr_3 has destination position $((e_1, 0\,\text{m}, 300\,\text{m}))$.

The *movement* m of a train tr is described by a range such that $l_{tr} \leq l_m \leq l_{tr} + s_{tr}$. Intuitively, a single movement is described by the parts of the track in the network the train moves over in one time step. Because of that, a movement of a train is at least as long as the train itself (even if a train stands still in one time step, it still covers the part of the network its standing on). Furthermore, a movement covers the train's position before and after the movement. The direction of a movement is implicitly given by the sequence of TTD intervals. Then, a *route* R_{tr} of a train tr can be described by a sequence of movements $(m_1, \ldots, m_n)$ such that m_1 starts at $start_{tr}$ and m_n ends at end_{tr}. The *travel time* $|R_{tr}| = n$ of R_{tr} is the number of time steps it takes for train tr to reach its goal. The problem considered in this work is to determine routes of all trains $tr \in Tr$ in the given railway network G such that, e. g., the sum of travel times of all trains is minimal.

Note that there might be additional constraints on movements and routes imposed by the underlying railway network. Because TTD points may represent hardware such as switches, a train might not be able to move from one edge to another even if they are connected via a TTD point. In Figure 1, for example, trains cannot move from e_3 to e_6. Also trains can of course not just change directions during a route.

Example 3. Consider again the layout and specification of trains given in Fig. 1. Figure 2 shows example movements for all trains in the network if we assume a time step of 15 s. The movement m_1 of train tr_1 is given by the range consisting of two TTD intervals $m_1 = ((e_2, 200\,\text{m}, 100\,\text{m}), (e_6, 0\,\text{m}, 600\,\text{m}))$. Similarly the movement m_2 is given by the range consisting of two TTD intervals $m_2 = ((e_1, 100\,\text{m}, 500\,\text{m}), (e_2, 0\,\text{m}, 400\,\text{m}))$. Train tr_3 remains at the same position. This is still considered to be a movement which is simply the position of the train (therefore, m_3 is set to $((e_6, 650\,\text{m}, 950\,\text{m}))$). Note that the lengths of the movements might be smaller than the length of the respective train.

Solving this problem for a single train is relatively simple (as reviewed later in Section 4.1). However, determining the fastest routes becomes significantly harder if multiple trains need to be considered because, then, collisions have to be avoided. More precisely, there are two types of collisions: (1) a *move collision* occurs if two movements overlap and (2) a *TTD collision* occurs if two movements contain the same TTD and no move collision occurs. Two routes R_1 and R_2 are then in collision if any of its movements are in collision.

Example 4. Consider again the movements depicted in Fig. 2. A move collision occurs between movements m_1 and m_2 since both movements overlap on the range $((e_2, 20\,\text{m}, 40\,\text{m}))$. A TTD collision occurs between movements m_1 and m_3 since both movements contain a TTD interval on TTD e_6.

Collisions substantially harden the railway routing problem and frequently lead to solutions with overly long travel times. Thus far, move collisions have been avoided by re-routing trains until no more move collisions occur. But using modern railway systems such as ETCS Level 3, CTCS Level 3+/4, or TPWS, TTD collision can also be resolved by introducing virtual subsections, i. e., VSS. Recall that a TTD collision only occurs when two trains occupy the same TTD, but would not occur if there would be a TTD point between the two trains. VSS basically introduces such (virtual) TDD points and, by this, can help resolving these TTD collisions without the need to re-route trains[3].

Based on all that, the railway routing problem considered in this work can be succinctly described as: Given a railway network $G = (V, E, L)$ as well as a

[3] Note that, for our purposes, blocks defined by TTDs and blocks defined by VSS are indistinguishable. We can therefore interpret a *VSS layout* of a railway network G as a graph G' that is obtained from G by splitting TTDs into VSS. This partitioned graph then simply defines a new railway network. There are in general an infinite number of VSS layouts for a given railway network. We will see in Section 4 how to handle this search space.

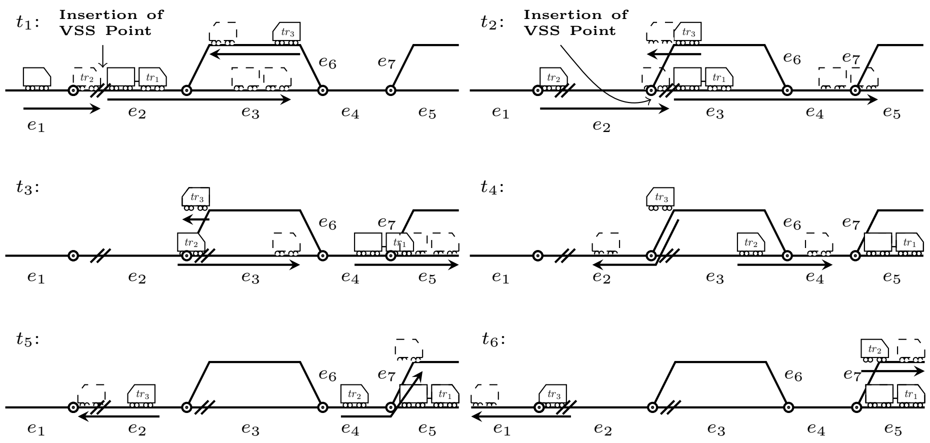


Fig. 3: Possible routes for Example 5

set of trains Tr with start and destination positions, determine a VSS layout $G' = (V', E', L')$ and a set of routes $\{R_{tr} \mid tr \in Tr\}$ on G' such that the objective $\sum_{tr \in Tr} |R_{tr}|$ (i. e., the sum of travel times) is minimized.

Example 5. Let's consider the layout specified in Fig. 1 again and assume time steps of 15 s as in Example 3. Possible routes for the three trains are shown in Fig. 3. Train tr_1 reaches its destination in time step t_3, trains tr_2 and tr_3 reach their destinations in time step 6. The sum of travel times is therefore 15 time steps, i. e., 225 s. In time steps t_1 and t_2, a TTD collision occurs between tr_1 and tr_2. These are repaired by introducing a VSS in TTD e_2 and e_3. Without these VSS, tr_2 would have to wait until tr_1 has left e_2 completely before it can enter. This would delay the route of tr_2 and tr_3 (because it has to wait until tr_2 has left e_2) by one time step. Hence, the introduced VSS indeed improved the train movements.

3 Motivation: The Problem of Discretization

Virtual subsections as introduced above provide a huge degree of freedom that allow for a more efficient railway routing. At the same time, they make the task of determining the best possible routes substantially harder. Because TTDs can be split up into VSS at arbitrary positions in the network, the resulting VSS layouts can be very complex. Techniques to solve the routing problem while simultaneously generating VSS layouts are still in its infancy. An existing solution tackling this problem is proposed in [25] where the problem is defined as a satisfiability problem and handed over to a reasoning engine. In that work, we model the search space of VSS layouts by discretizing the network. This is needed to model the possible positions of the trains on the network. While there are in theory an infinite number of positions, the discretization narrows this search space down such that it can be modelled using a finite number of Boolean variables.

To this end, a *spatial resolution* r_s is defined. Every TTD is then split into *segments* of length r_s. That is, with a spatial resolution of $r_s = 100\,\text{m}$, a TTD of length 1 km would be split into 10 smaller segments. Such a discretized TTD can then no longer give rise to arbitrarily many VSS, rather VSS can only be composed of these segments.

Discretizing the network like this does not only simplify the search space for VSS layouts, but also the train movements. Similarly to VSS, train positions are described by the segments that are occupied by a train. For example, the trains in Fig. 4 occupy one and two segments, respectively. The trains might actually be much shorter than r_s but they are still considered to occupy an entire segment. Train speeds are therefore defined in terms of *segments traversed per time step*, i. e., a train *tr* with speed s_{tr} has a *discretized speed* of $s_{tr} \cdot \frac{r_t}{r_s}$, where r_t is the so-called *temporal resolution*, the duration of one time step. For example, with $r_t = 10\,\text{s}$ and $r_s = 100\,\text{m}$ a train with a speed of 200 km/h would have discrete speed of about 5.556.

Although this discretization yields a smaller search space, it also causes several problems.

- *Infeasible configurations:* Because train movement is described in terms of segments traversed per time step, the choice of temporal resolution depends on the spatial resolution. A fine temporal resolution combined with a coarse spatial resolution can lead to situations where trains can seemingly not move at all or move faster than they should be able to.
- *Rounding Errors:* An improper discretization can lead to incorrect solutions. Because the simulation proceeds in discrete steps, train speeds can only be integral values. Therefore sub-optimal routes may be found when speeds are rounded down; or impossible solutions may be found when speeds are rounded up. Even if constraints are added enforcing that resolutions must be chosen such that train speeds can be modelled accurately, this constraint becomes harder and harder to satisfy the more trains are to be considered.
- *Oversimplifications:* Imposing a spatial resolution leads to difficulties when finer details of the network should be accurately portrayed. This can be mitigated by choosing different resolutions for different parts of the network but opting to do this again increases the complexity of the network.

In addition to these issues the approach from [25] also requires the definition of the maximum number of time steps the trains can take to ensure a finite search space. All of the above combined makes it very hard for a designer to choose correct configurations in order to obtain a good solution. Controlling for all issues at the same time is highly non-trivial or might even be infeasible. It is also difficult to judge the quality of a found solution. Infeasible configurations and oversimplifications can lead to solutions that are better than what is possible in reality, whereas rounding errors lead to solutions that are worse than the “real” optimum.

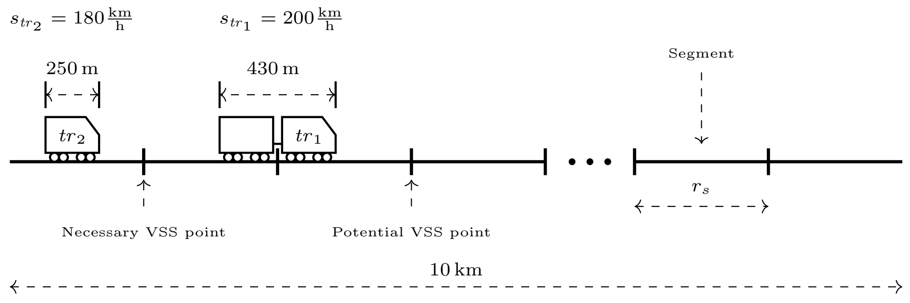


Fig. 4: Discretizing space

Example 6. Consider the simple layout shown in Fig. 4 consisting of a single TTD with a length of 10 km and 2 trains tr_1 and tr_2 moving from left to right with maximal speeds of 200 km/h and 180 km/h as well as lengths of 430 m and 250 m, respectively. Now, the following problems may emerge when trying to choose proper values for r_s and r_t.

Choosing $r_t = 10\,\mathrm{s}$ and $r_s = 1000\,\mathrm{m}$ leads to tr_1 having a discretized speed of 0.556. If we round down, then tr_1 would not be able to move at all. If we round up, tr_1 takes 10 time steps to reach the other end of the track. Converting this speed back to real units gives a speed of 360 km/h which does not remotely reflect the actual speed of tr_1.

Choosing $r_t = 10\,\mathrm{s}$ and $r_s = 100\,\mathrm{m}$ leads to discretized speeds 5.556 and 5 for tr_1 and tr_2, respectively. Here, speeds are rounded down and, thus, tr_1 is treated as having an actual speed of 180 km/h. It is easy to see that, with the chosen temporal resolution, the optimal number of time steps for tr_1 to reach the right end of the TTD is 18 time steps. But with the rounded speed, the best solution that can be found takes 20 time steps. Moreover, since both trains are treated as having the same speed, more VSS have to be placed than necessary. Since the lengths of the trains are not multiples of 100 m, they take up more space of the network than necessary. More specifically, tr_1 would occupy 5 segments at any time step. This prevents tr_2 from moving as close to tr_1 as possible, yielding a suboptimal route for tr_2.

All these problems may be avoided by choosing $r_t = 10\,\mathrm{s}$ and $r_s = 5\,\mathrm{m}$. Then, the trains have discretized speeds of 100 and 111 respectively, thus avoiding the impact of rounding errors as much as possible. But this would partition the TTD into 2000 segments, an unreasonably fine grained discretization.

These examples show that the problems described above already occur in very simple scenarios. Motivated by that, this work proposes an alternative approach that overcomes these problems by avoiding discretizing the network at all – while still being able to determine optimal railways routings.

4 Proposed Solution

This section describes the proposed alternative solution to the optimal railway routing problem described above. Its main approach rests on an A*-based search scheme which is described first. Afterwards, we explicitly describe how virtual subsections are utilized to resolve collisions. Using both, A* and the extended degree of freedom through those VSS eventually allow to generate optimal railway layouts.

4.1 Main Approach Based on A* Search

A* Search is a state-space search algorithm. That is, the search space is defined over states $s \in S$ (with S being the set of all possible states) and transitions between them. Starting from an initial state, the goal is to determine a route towards a goal state within that search space which satisfies a certain goal condition. By dedicated functions, the total costs of the current state are tracked while the remaining costs towards the goal state are estimated. By this, A* Search traverses through the search space – ideally only expanding towards states with the lowest cost and, by this, avoiding traversing parts of the search space that lead to no or overly expensive solutions.

In order to solve the problem reviewed in Section 2, we use the main concept of A* Search as a basis. More precisely, given a set of trains Tr, we model a state $s \in S$ at time step t as the set of positions of all trains, i. e., $s = \{pos_{tr}^{t_{tr}} \mid tr \in Tr\}$, where t_{tr} is the time step at which train tr has reached its destination or t if tr has not reached its destination yet. Two states s and s' are then connected if all trains can make a movement from their position in s to their position in s' within one time step and without causing any collisions. The initial state is then $s_{init} = \{start_{tr}^{0} \mid tr \in Tr\}$ and the goal states are defined by $s_{goal} = \{dest_{tr}^{t_{tr}} \mid tr \in Tr\}$.

Because trains do not move in discrete steps, there is an enormous number of successors for each state. While this is true in principle, most of these positions are redundant. The only branching points in the network are at TTD points. It is therefore superfluous to consider every possible position a train might have within a TTD and consider only movements that transport a train as far as possible within a TTD.

Example 7. Let's consider the start positions in Fig. 1 as an example for an initial state s_{init}. The successor states of s_{init} are derived by considering all possible combinations of movements the three trains in this example can make. As discussed previously, only those movements are considered that move the trains as far as possible within one TTD. In this example, there are 6 potential successor states $s_i, 1 \leq i \leq 6$, as shown in Fig. 5.

However, states s_3 and s_6 are not valid, because the movements of tr_1 and tr_3 are in collision (as indicated by $\lightning$ in Fig. 5). Similarly, s_4 and s_5 are not valid, because tr_1 and tr_2 are in collision. Therefore, s_{init} has only two real successor states, namely s_1 and s_2.

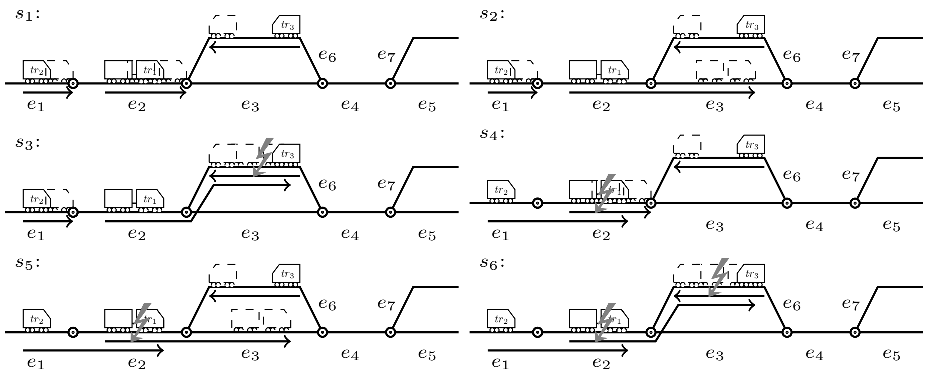


Fig. 5: A* search

Recursively or iteratively searching through *all* possible states eventually would lead to several goal states. Out of those, we are then interested in the one with the smallest costs. These are provided by means of a function $g(s)$ which gives, for a state s, the costs up to this state. In our case, the costs are defined by the sum of total travel times[4], i. e., for state $s = \{pos_{tr}^{t_{tr}} \mid tr \in Tr\}$ we have $g(s) = \sum_{tr \in Tr} t_{tr}$.

However, to avoid expanding towards non-promising states (i. e., states that will not lead to a goal state or only through substantially longer paths), a *heuristic function* $h(s)$ is additionally employed. This heuristic function assigns to each state s an *estimation* of the costs from s towards a goal state. If we knew the distances d_{tr}^{s} from the position of each train tr in state s to its destination position $dest_{tr}$, those costs can be estimated by $h(s) = \sum_{tr \in Tr} \frac{d_{tr}^{s}}{s_{tr}}$. These distances can easily be obtained for a train tr by precomputing a lookup table of distances from each TTD to the destination of tr via breadth-first search. To get the true distance from this lookup table, the offset of the train within the TTD has to be subtracted. Overall, this leads to a total costs of a state $s \in S$ defined as $f(s) = g(s) + h(s)$.

An important property of A* Search is that an optimal solution is guaranteed to be found if the heuristic function is *admissible*. A heuristic function is admissible if the heuristic function never overestimates the true cost to a goal state. It is easy to see that the heuristic function defined for our problem is admissible since trains can never arrive at their respective destinations faster than if they were traveling with maximum speed for the entire route.

Example 8. Consider again the example in Fig. 5. There are two possible successors of s_{init}, s_1, and s_2. In s_2, all trains are closer to their goal than in s_1. Therefore, the estimated sum of travel times is smaller in s_2 and, hence, s_2 would be expanded next by the search.

[4] Note that the cost function can, of course, accordingly be adjusted if the focus is put on other aspects such as the overall travel time.

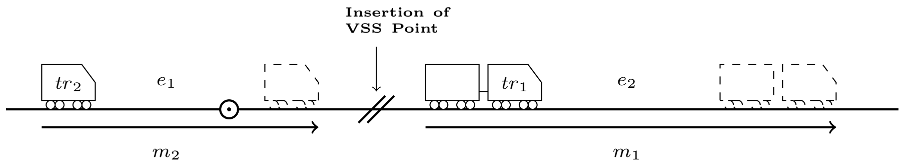


Fig. 6: Resolving TTD collisions with VSS

4.2 Resolving Collisions with VSS

Using A* Search as proposed above, we are looking for a path in the search space from the initial state s_{init} to one of the goal states s_{goal} with the smallest costs. But, thus far, possible paths are severely restricted since states are connected only if trains can reach them without causing any collisions. Using VSS, however, many collisions can be prevented – likely leading to faster routes. Recall from Section 2 that there are two types of collisions: move collisions and TTD collisions. Let's consider TTD collision first because resolving them is conceptually easier. A TTD collision occurs when two movements contain the same TTD but are not in a move collision. This kind of collision can be resolved by splitting the TTD into two separate VSS.

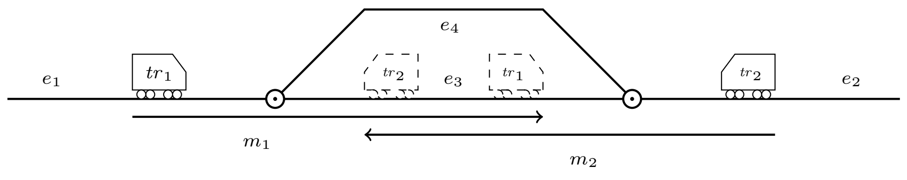
Example 9. Consider the situation in Fig. 6 involving two trains moving on a straight track. Movements m_1 and m_2 are in a TTD collision on TTD e_2. By splitting e_2 into two VSS at the point shown in the figure, the TTD collision is resolved.

For move collisions, we can identify three different cases:

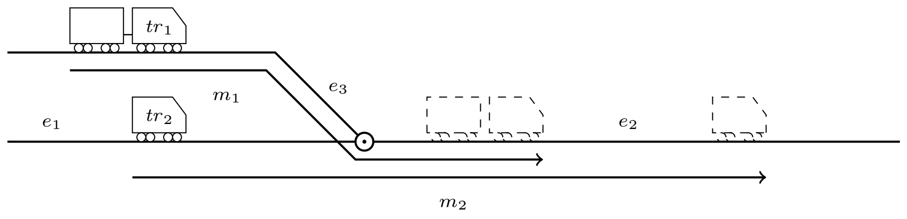
- A *head-on-collision* occurs when two movements go in opposite directions in a TTD.
- An *overlap collision* occurs when two movements go in the same direction in a TTD that is not the start of either movement.
- A *rear-end-collision* occurs when the end of a movement collides with the start of another movement.

Head-on-collisions and overlap collisions can not be resolved. In these two cases, corresponding states cannot be connected and, hence, the A* Search needs to determine alternative routes. But rear-end-collisions can be resolved: If movement m_1 rear-end-collides with movement m_2, then a new movement m_1' can be obtained by truncating m_1 in such a way that no collision with m_2 occurs anymore. The movements m_1' and m_2 are then in a TTD collision. But as we have seen previously, these can be resolved.

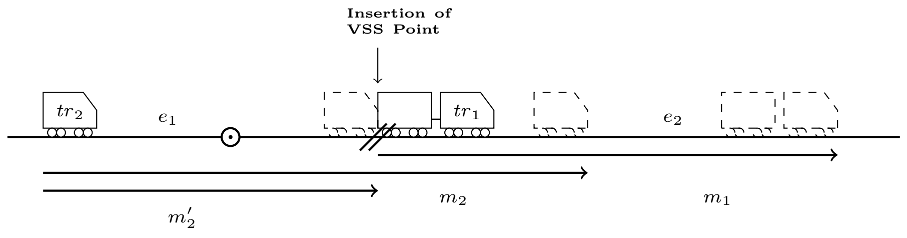
Example 10. The three different types of movement collisions are depicted in Fig. 7. Figure 7a is an example of a head-on-collision. It is apparent that this situation cannot be rectified by introducing further VSS.



(a) Head-on-collision



(b) Overlap collision



(c) Rear-end-collision

Fig. 7: Move collision cases

Figure 7b is an example of an overlap collision. As it was in the case of a head-on-collision, the collision cannot be resolved by splitting TTDs into VSS. The conflict can only be avoided if one of the trains waits with their movement, possibly leading to a longer route.

Figure 7c is an example of a rear-end-collision. Movement m_1 and m_2 are in collision on TTD e_2. But by moving tr_2 with movement m_2' instead of m_2 the collision is avoided.

5 Experiments

The A*-based approach as described above has been implemented in C++ and was subjected to several benchmarks to evaluate its performance. Additionally, we also considered the solution described in [25] in order to compare the proposed solution to the current state of the art. Both implementations are part of the DA-ETCS toolkit available at `https://iic.jku.at/eda/research/etcs/`. All those experiments have been conducted on an Intel(R) Core(TM) i7-7700K machine using a 4.20 GHz processor with 32 GB of main memory running Ubuntu 18.04.4. In this section, the obtained results are summarized and discussed.

As benchmarks, we considered the railway layouts and tasks which have been used in the evaluations in [25] (namely, *Running Example*, *Simple Example*, *Complex Example*, and *Nordlandsbanen*) as well as further instances of representative use cases (namely, *Bottleneck*, *Bidirectional*, and *Train Station*). Here, *Bottleneck* refers to a track layout where all trains have to be funneled through a single TTD before their paths diverge again, while *Bidirectional* refers to a layout where trains are moving in both directions on a main track with occasional sidings to pass each other. Finally, *Train Station* refers to a track layout with several interconnected and branching paths for the trains to reach their goals. All these benchmarks represent frequent use cases that usually have many potential collisions to be avoided through the use of virtual subsections.

The obtained results are shown in Tab. 1. Here, for each considered benchmark, the generated results for both methods (as indicated in the first column) are provided in the respective lines of the table. As described in Section 3, the approach from [25] always needs a *configuration* in terms of the spatial resolution r_s and the maximum number of time steps t_{max} in order to generate a discrete formulation; these values are provided in the second and third column. Afterwards, the respectively obtained results are presented, i. e., the number blocks (both, TDDs and VSS combined), the number of *Time Steps* until all trains have reached their goal, and the sum of travel times of all trains $\sum t$ (the actual optimization objective which has been optimized). Finally, the required runtime is provided (note that, in case of the approach from [25], only the solving time is listed, even though also the runtime for generating the discrete formulation often is substantial).

The obtained results clearly confirm the shortcomings of the previously proposed approach as discussed in Section 3 and show how the approach proposed in this work addresses them. More precisely:

First, since the approach from [25] always requires a discrete formulation, the designers are urged to provide a configuration in terms of spatial resolution r_s and maximal number of timesteps t_{max}. This frequently pushes him/her to trade-off between accuracy (demanding a finer resolution) and performance (demanding a coarser resolution). These values require prior knowledge about the benchmark like an estimation of the time steps a solution might have. If no proper estimation can be made, multiple configurations have to be tried such that a satisfactory solution can be obtained in an iterative fashion (which is why, we present several configurations in Tab. 1). In contrast to all that, the proposed approach does not require a configuration and does not rely on a discrete formulation, which is why all these problems do not occur here.

Second, the precision of the approach from [25] highly depends on the respectively chosen configuration (and, hence, discretization). This explains the huge differences in the obtained results. In the worst case, choosing an improper configuration may yield a formulation out of which no solution can be generated at all. This is the case in the two instances marked *Unsatisfiable* where the maximum number of time steps is too small to allow a solution to be found with the given discretization (although the optimum from the A* Search shows that a solution indeed is possible). But even if solutions are determined, they are often significantly off and, hence, imprecise compared to the actual optimal value (obtained by the A* Search without discretization). All this basically confirms the discussions from Section 3 about the shortcomings of the discretization and shows that the A* Search proposed in this work nicely addresses these problems.

Finally, the runtime performance of both approaches confirms what could be expected. The coarser the resolution and, hence, discretization of the approach from [25], the better its runtime. In order to get precise results, however, this frequently leads to timeouts (in our evaluations of 1 hour). In contrast, the proposed method's main drawback is an increased memory requirement – in particular in cases where the A* Search expansion leads to a huge number of possible states to consider (as in the case of the last instance in Tab. 1). These cases, however, usually also cannot be handled by the approach from [25] (at least, not with proper precision) and most likely constitute instances, where optimal railway routing probably reaches its limits due to the underlying complexity (in the worst case, both methods exhibit exponential time or space complexity). In all other cases, the A* Search clearly outperforms the state of the art and often yields magnitudes of better runtime – in particular compared to instances with proper precision.

6 Conclusion

In this work, we considered the automatic generation of optimal railway routings for modern railway systems such as the ETCS Level 3, CTCS Level 3+/4, or TPWS, which allow for virtual subsections. To this end, we first analyzed the major shortcomings, namely infeasible configurations, rounding errors, and oversimplifications, of the current state of the art which are mainly caused by discretization. We proposed an approach which addresses all these problems and, at the same time, even led to substantial runtime improvements (reaching several orders of magnitudes). Experiments and detailed comparisons confirmed these benefits.

Acknowledgments

This work has partially been supported by the BMK, BMDW, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).

References

1. J. Pachl, *Besonderheiten ausländischer Eisenbahnbetriebsverfahren*, 2019.
2. ——, *Railway Signalling Principles*, Braunschweig, Jun 2020.
3. "Set of specifications 1/2/3," https://www.era.europa.eu/content/ccs-tsi-annex-mandatory-specifications, accessed: 2020-09-18.
4. P. Stanley and I. of Railway Signal Engineers, *ETCS for Engineers*, 2011.
5. Z. Yang, "Application and development of ctcs," *UIC ERTMS WORLD CONFERENCE*, vol. 12, 2016.

6. R. A. India, “Implementation of etcs and tpws system over indian railway network,” *Rail Analysis India*, 2019.
7. D. Dghaym, M. Poppleton, and C. Snook, “Diagram-led formal modelling using iUML-B for Hybrid ERTMS Level 3,” in *ABZ*, 2018.
8. D. Dghaym, S. Dalvandi, M. Poppleton, and C. Snook, “Formalising the hybrid ERTMS level 3 specification in iUML-B and Event-B,” *International Journal on Software Tools for Technology Transfer*, vol. 22, Jun 2020.
9. A. Cunha and N. Macedo, “Validating the Hybrid ERTMS/ETCS Level 3 concept with Electrum,” in *ABZ*, 2018.
10. S. J. Tueno Fotso, M. Frappier, R. Laleau, and A. Mammar, “Modeling the Hybrid ERTMS/ETCS Level 3 standard using a formal requirements engineering approach,” in *ABZ*, 2018.
11. J.-R. Abrial, “The ABZ-2018 case study with Event-B,” in *ABZ*, 2018.
12. A. Mammar, M. Frappier, S. J. Tueno Fotso, and R. Laleau, “An Event-B model of the Hybrid ERTMS/ETCS Level 3 standard,” in *ABZ*, 2018.
13. P. Arcaini, P. Ježek, and J. Kofroň, “Modelling the Hybrid ERTMS/ETCS Level 3 case study in Spin,” in *ABZ*, 2018.
14. Hoang, Thai Son and Butler, Michael and Reichl, Klaus, “The Hybrid ERTMS/ETCS Level 3 case study,” in *ABZ*, 2018.
15. D. Hansen, M. Leuschel, D. Schneider, S. Krings, P. Körner, T. Naulin, N. Nayeri, and F. Skowron, “Using a formal B model at runtime in a demonstration of the ETCS Hybrid Level 3 concept with real trains,” in *ABZ*, 2018.
16. J. Jansen, E. Quaglietta, M. Bartholomeus, A. Pot, and R. Goverde, “ETCS Hybrid Level 3: A simulation-based impact assessment for the Dutch railway network.”
17. D. Gill, “ETCS Level 3 for metro-type mainline operation,” in *Aspect*, 2017.
18. K. Ghoseiri, F. Szidarovszky, and M. J. Asgharpour, “A multi-objective train scheduling model and solution,” *Transportation Research Part B: Methodological*, vol. 38, no. 10, pp. 927–952, 2004.
19. X. Cai and C. Goh, “A fast heuristic for the train scheduling problem,” *Computers & Operations Research*, vol. 21, no. 5, pp. 499–510, 1994.
20. X. Zhou and M. Zhong, “Bicriteria train scheduling for high-speed passenger railroad planning applications,” *European Journal of Operational Research*, vol. 167, no. 3, pp. 752–771, 2005, multicriteria Scheduling.
21. C. Liebchen, “The first optimized railway timetable in practice,” *Transportation Science*, vol. 42, no. 4, pp. 420–435, 2008.
22. R. Lusby, J. Larsen, D. Ryan, and M. Ehrgott, “Routing trains through railway junctions: a new set-packing approach,” *Transportation Science*, vol. 45, no. 2, pp. 228–245, 2011.
23. J.-W. Goossens, S. van Hoesel, and L. Kroon, “On solving multi-type railway line planning problems,” *European Journal of Operational Research*, vol. 168, no. 2, pp. 403–424, 2006, feature Cluster on Mathematical Finance and Risk Management. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221704003169
24. G. Garrisi and C. Cervelló-Pastor, “Train-scheduling optimization model for railway networks with multiplatform stations,” *Sustainability*, vol. 12, no. 1, p. 257, 2020.
25. R. Wille, T. Peham, J. Przigoda, and N. Przigoda, “Towards automatic design and verification for level 3 of the european train control system,” in *Design, Automation and Test in Europe (DATE)*, 2021.

Table 1: Obtained results

Method	Configuration r_s [m]	t_{max}	TTD/VSS	Time Steps	$\sum t$	Runtime [s]
Running Example (with 4 trains an total travel length of 7 km)						
Approach from [25]	500	11	5	7	23	0.1
A* Search	-		9	7	21	< 0.1
Simple Example (with 4 trains and total travel length of 27 km)						
Approach from [25]	500	20	14	15	53	29.2
A* Search	-		26	15	50	< 0.1
Complex Example (with 6 trains and total travel length of 148 km)						
Approach from [25]	1000	18	25	16	71	124.9
A* Search	-		42	14	58	138.3
Nordlandsbanen (with 3 trains and total travel length of 819.6 km)						
Approach from [25]	1000	140	-	-	-	> 3600
A* Search	-		519	135	286	45.713
Bottleneck (with 4 trains and total travel length of 10 km)						
Approach from [25]	1000	20	13	18	60	0.6
	500	20	13	18	60	2.3
	100	20	16	15	54	84.9
	50	20	16	15	54	777.9
	50	15	16	15	54	866.5
A* Search	-		39	15	50	< 0.1
Bottleneck (with 10 trains and total travel lengh of 2.6 km)						
Approach from [25]	1000	20		*Unsatisfiable*		1185.9
	1000	30	-	-	-	> 3600
	100	15	-	-	-	> 3600
A* Search	-		30	12	65	11.1
Bottleneck (with 12 trains and total travel length of 3 km)						
Approach from [25]	1000	20		*Unsatisfiable*		1275.1
A* Search	-		34	15	92	371.0
Bidirectional (with 6 trains and total travel length of 14.6 km)						
Approach from [25]	1000	30	16	30	124	50.6
	500	30	18	21	112	698.2
	100	30	-	-	-	> 3600
	100	23	-	-	-	> 3600
A* Search	-		53	22	105	1.6
Train Station (with 6 trains and total travel length of 7.1 km)						
Approach from [25]	1000	30	19	9	39	1.1
	500	30	19	9	39	1.1
	100	30	31	21	114	64.1
	50	30	31	22	117	1381.3
A* Search	-		58	22	110	17.7
Train Station (with 8 trains and total travel length of 7.3 km)						
Approach from [25]	1000	30	21	11	59	9.6
	500	30	21	11	59	9.6
	100	30	-	-	-	> 3600
	100	23	33	23	159	564.1
A* Search	-			*Out of Memory*		-