Abstract—Railway capacity is complex to define and analyze, and existing tools and methods used in practice require comprehensive models of the railway network and its timetables. Design engineers working within the limited scope of construction projects report that only ad-hoc, experience-based methods of capacity analysis are available to them. Designs have subtle capacity pitfalls which are discovered too late, only when network-wide timetables are made – there is a mismatch between the scope of construction projects and the scope of capacity analysis, as currently practiced.

We suggest a language for capacity specifications suited for construction projects, expressing properties such as running time, train frequency, overtaking and crossing. The planning problem arising from verification of these properties is constrained by discrete control system logic, network topology, laws of motion, and sparse communication. The second-order linear differential equations describing train motion can be solved analytically, but the resulting expressions contain multiplication of real variables.

We argue that reasoning over the whole discrete/continuous space of solutions is not efficient with current state-of-the-art solvers, and we present instead a special-purpose SAT-based dispatch planning tool which solves the discrete part of the problem. Continuous dynamics and timing constraints are evaluated using discrete event simulation, and the two components communicate in a CEGAR-loop (counterexample-guided abstraction refinement). We show that our method is fast enough at relevant scales to provide agile verification in a design setting, and we present case studies based on data from existing infrastructure and ongoing construction projects.

I. INTRODUCTION

This paper addresses a central problem that occurs when designing the layout and control systems for railway stations: In order to know whether or not the station can perform its intended functions, the designer needs to make an assessment of some notion of capacity.

As an example, consider the question of crossing trains on a railway station. Fig. 1 shows two sequences of movements which result in such a crossing. There are a number of details of the railway design which can cause this scenario to become infeasible (or take an unacceptably long time), such as signal placement, detector placement, correct allocation and freeing of resources, track lengths, train lengths, etc.

However, systematic capacity analysis for railways is typically performed on the scale of national railway networks, using comprehensive input on infrastructure and timetables, and only after the complete design is finished. Moreover, the widely used methods and tools for capacity analysis are

Fig. 1: Two alternative plans for achieving a crossing of two trains on a two-track station. The green areas show track segments which are currently occupied by a train going from left to right, while the pink areas show track segments which are currently occupied by a train going from right to left.

heavy-duty methods, consisting of complicated simulations, and require specialized knowledge, thus not being suitable for agile design-time verification of railway stations. As a consequence, railway construction projects usually rely on informal, vague, or even non-existent capacity specifications, and engineers need to make ad-hoc/manual analyses of how the control system can provide this capacity.

Our goal is to develop verification techniques and tools to help engineers at design time specify capacity properties and to check these automatically. To be agile, the tool needs to (1) have reasonable running times so that the verification can be run on the fly as the design is being updated by an engineer working in a drafting CAD application, and (2) keep the required input to the minimum of information needed to verify relevant properties. This style of verification gives engineers immediate feedback on their design decisions while requiring small amounts of specification and verification work.

The problem: We seek to solve the low-level railway infrastructure capacity verification problem, which we define as follows:

Given a railway station track plan including signaling components, rolling stock dynamic characteristics, and a performance/capacity specification, verify whether the specification can be satisfied and find a dispatch plan as a witness to prove it.

Solving this problem subsumes the following railway infrastructure design activities:

- Low-level running time analysis – verify the time required for getting from point A to point B.
- Low-level **schedulability** analysis – verify frequency of trains arriving at a station, and simultaneous opportunities for crossing, parking, loading, etc.
- **Combinations** – verify running time requirements on schedulable operations.

**Our approach:** In this paper we suggest a formalization of capacity requirements as a set of operational scenarios involving a set of trains, a set of locations to visit, and a set of timing constraints.

Verification in this domain can in principle be encoded into the SMT [1], [2] or PDDL+ [3] languages, essentially resulting in a SAT modulo non-linear real arithmetic problem [4]. Many solvers can handle such problems, but we found that the problem size of our test cases, in terms of the number of planned actions and in terms of number of interacting Boolean and non-linear real logic terms, were out of reach for agile verification. Also, train dynamics using only constant acceleration \( x'' = c \) is in some cases too simplistic for engineering. We would like to be able to extend the dynamics equations using e.g. polynomials of higher order or even numerical integration.

Therefore, we have developed a verification tool chain that uses a CEGAR-loop between a SAT-based planning tool that works on a discrete abstraction of control system commands, and a discrete event simulation engine (DES) [5] that calculates detailed continuous results for a specific plan, taking the physics of moving trains into account.

The SAT-based planner uses bounded model checking (BMC) [6] where time is reduced to a series of partially ordered actions with unknown durations, and the choice of actions are the available commands in the control system. The DES component verifies the continuous time/space results given the Boolean decisions of control system commands, and adds new SAT constraints excluding unsatisfactory solutions.

The separation of discrete and continuous domains also has the advantage that the simulation component can be extended to handle more complex models, such as engine power curves, tunnel air resistance, curve rolling resistance, train weight distribution, etc. without affecting the planning logic or its computational complexity.

We have tested our method and tool on practical examples from existing infrastructure and ongoing construction projects in collaboration with railway engineers in Railcomplete AS.

The rest of the paper is organized as follows: Section II contains an overview of the railway design process and the principles for analysis of these designs. We present a structure for capacity specifications, together with examples of how they can be used in construction projects. Section III describes the tool chain and the solver architecture that we propose to verify performance properties and integrate agile verification in the construction project workflow, and how each of the components of our solver are implemented. Section IV contains performance evaluations in a set of relevant case studies. Section V gives pointers to related work, and Section VI presents our conclusions.

II. Domain Background and Problem Description

Railway capacity is hard to define precisely (see [7] for a discussion). Any capacity measure will necessarily make assumptions about the operation of the railway. One can say that the railway infrastructure does not have an inherent capacity, only capacity for specific use cases. As such, a fully accurate assessment of capacity can only be made under a fully specified timetable, meaning that every train’s arrival and departure times at all stations in the network must be known. This makes for a highly coupled analysis, as constructing an actual timetable requires bringing together details about infrastructure, rolling stock, transportation demands, and crew schedules. Such work can be done using commercial tools like RailSys [8], OpenTrack [9], LUKS [10], and others. See [11] or [12] for an overview of methods.

The so-called analytical approaches to capacity analysis using networked queuing theory [13], maximum flow (originally posed as a railway capacity problem [14]), or max-plus algebra [15], can give preliminary or low-precision network-wide results, but fail to account for the critical low-level factors which are relevant for verification in construction projects, specifically discrete control system logic, communication, and train acceleration and braking dynamics.

Because the verification feedback loop between design and capacity analysis is either very time-consuming or too coarse-grained, railway engineers end up re-using proven design concepts or allowing sizable margins, e.g. in track lengths.

However, modern construction practice expects and demands optimization. When space requirements, performance requirements and costs are squeezed to the limit, the tradition-based railway engineering approach lacks the methods to accurately reason about the expectations of the finished system from partially finished design plans.

Using agile verification of high-level properties from the beginning of a design project, and in every step of the process, allows engineers to better see the consequence of each decision, and immediately uncover errors and shortcomings that would otherwise be discovered only months or years later.

**Railway design**

The railway design activity produces the following artifacts:

- Track and trackside component layout, describing the locations of tracks, switches, signals and detectors (see Fig. 2a).
- Interlocking specifications, describing the requirements for the logic of the control system (see Fig. 2b).

These design artifacts are the subject of verification, i.e. the model. Ensuring performance in the context of a construction project consists of verifying properties describing a set of trains moving on the tracks and the goals which need to be accomplished by these movements.

To verify performance properties, we need a to find a sequence of trains and elementary routes for the train dispatcher, i.e., a dispatch plan, which when executed under safety and
A. Safety and correctness of train movements

Low-level analysis of train movements covers a wide range of constraints given by the track layout, the control system, and operational procedures, to be certain that the analysis produces detailed, realistic results. The following subsections give an overview of these constraints, divided into four classes.

1) Physical infrastructure: Trains travel on a network of railway tracks which have physical properties such as length, gradient, curvature, etc. Tracks branch off using switches, whose setting determines where the train goes. Detectors on the track are used by the control system to determine whether track segments are occupied. The physical infrastructure also determines the sight areas: the set of locations where a train receives information from a given signal.

2) Allocation of resources: Avoiding collisions by exclusive use of resources is the responsibility of the interlocking, which takes requests for activating elementary routes from the dispatcher. Elementary routes are smallest unit of resources that can be allocated to a train, see Fig. 3. Route activation is a process which proceeds as follows:

1) Wait for all required resources, such as track segments and switches, to be free. Resources required by a route are typically any resource in the train path (or sometimes outside of it), which ensure that all movements are performed at a safe distance from each other.

2) Movable elements (e.g. switches) must be set to correct positions. If they are not, start a sub-process which moves the element into place, and wait for this process to finish before proceeding.

3) Signals are then set to show the ‘proceed’ aspect to the train when the above steps are finished. When the front of the train has passed the signal, it is immediately reset to show the ‘stop’ aspect.

4) A release process is started, which waits for the train to finish using the allocated resources (i.e. to travel over them) and frees them when this has happened.

3) Communication constraints: After movement has been allowed by the control system, the driver must be informed of this fact. When a route is activated, a train inside the sight area of the route’s entry signal reads the signal’s message that movement authority is given. The train driver may then drive the train forward until the next signal. The following types of signalling systems are common in railways:

- Traditional signaling with trackside lamps. Communication is limited by how many different aspects the lamps can show. To avoid high-speed trains slowing down at every signal, several consecutive elementary routes can be signaled in advance using so-called distant signals.

- Automatic train protection systems (ATP) work similarly to signals, but may give more information. Many ATP systems communicate information through magnets or short-range radio at specific locations on the track, corresponding to a signal sight area of zero length.

- The European Rail Traffic Management System (ERTMS) currently being implemented in many European countries replaces lamp signals with trackside marker boards, and uses long-range radio for communication. This effectively removes the communication constraint, as the radio can be used to update any train’s movement authority at any time.

4) Laws of motion: Trains move within the limits of given maximum acceleration and braking power. Train drivers need to plan ahead for braking so that the train respects its given movement authority and speed restrictions at all times.

The speed increase from $v_0$ to $v$ over a time interval $\Delta t$ is limited by the train’s maximum acceleration $a$:

$$v - v_0 \leq a\Delta t.$$ 

However, when there is a more restrictive speed restriction ahead, the driver must start braking in time to meet the restriction. A signal showing the ‘stop’ aspect can be treated as a speed restriction of zero. Since speed restrictions change
with time, the driver must re-evaluate their actions whenever new information is received.

A train has the following constraint on its velocity \( v \) for each restriction,

\[
v^2 - v_i^2 \leq 2bs_i,
\]

where \( v_i \) is the maximum allowed speed, \( s_i \) is the distance to the location where the restriction starts, and \( b \) is the maximum retardation achieved by braking.

See [16] for a more in-depth description of railway operation principles.

B. Station performance requirements

To capture typical performance and capacity requirements in construction projects, we define an operational scenario \( S = (V, M, C) \) as follows:

1) A set of vehicle types \( V \), each defined by a length \( l \), a maximum velocity \( v_{\text{max}} \), a maximum acceleration \( a \), and a maximum braking retardation \( b \).

2) A set of movements \( M \), each defined by a vehicle type and an ordered sequence of visits. Each visit \( q \) is a set of alternative locations \( \{i_l\} \) and an optional minimum dwelling time \( t_d \).

3) A set of timing constraints \( C \), which are two visits \( q_a, q_b \), and an optional numerical constraint \( t_c \) on the minimum time between visit \( q_a \) and \( q_b \). The two visits can come from different movements. If the time constraint \( t_c \) is omitted, the visit are only required to be ordered, so that \( t_{q_a} \leq t_{q_b} \).

To demonstrate how this structure captures requirements of railway construction projects, we give some examples using the syntax of the file format used in our tool[1]. First, we define the following vehicle types:

```plaintext
vehicle passengertrain length 220.0
   accel 1.0 brake 0.9 maxspeed 20.0
vehicle goodstrain length 850.0
   accel 0.5 brake 0.5 maxspeed 20.0
```

The following set of performance specifications are selected prototypical versions of specifications that railway engineers have suggested as useful for automated verification:

- Running time: expresses an expectation of how long it should take for a train to travel between two locations. To specify this, we simply require that a train visits some location \( b1 \) and later visits some other location \( b2 \). A timing constraint of 90.0s between these visits sets the running time requirement.

```plaintext
movement pasengertrain {
   visit #a [b1]; visit #b [b2] }
timing a < 90.0 b
```

- Train frequency: a train station processes a set of trains arriving and departing with a fixed frequency. On a two-track station, we exemplify a sequence of four trains and their relative departure times.

```plaintext
movement passengertrain {
   visit [b1]
   visit [platform1,platform2] wait 60.0
   visit #e1 [b2] }
// ...3 more trains with visits e2, e3, e4.
timing e1 < 90.0 e2
timing e2 < 90.0 e3
timing e3 < 90.0 e4
```

- Overtaking: trains traveling in the same direction can be reordered. For example, we specify a passenger train traveling from \( b1 \) to \( b2 \), and a goods train with the same visits. Timing constraints ensure that the passenger train enters first while the goods train exits first.

```plaintext
movement passengertrain {
   visit #p_in [b1]; visit #p_out [b2] }
movement goodstrain {
   visit #g_in [b1]; visit #g_out [b2] }
timing p_in < g_in
timing g_out < p_out
```

- Crossing: trains traveling in opposite directions can visit this station simultaneously. This example is similar to the previous one, but the goods train now travels in the opposite direction, and the timing constraints require that the trains are inside the model simultaneously.

```plaintext
movement passengertrain {
   visit #p_in [b1]; visit #p_out [b2] }
movement goodstrain {
   visit #g_in [b2]; visit #g_out [b1] }
timing p_in < g_out
timing g_in < p_out
```

Similar specifications, and combinations of such specifications, are relevant in most railway construction projects. Since we typically only need to refer to locations such as model boundaries and loading/unloading locations, these specifications are not tied to a specific design, and can often be re-used even when the design of the station changes drastically.

III. TOOL CHAIN AND SOLVER ARCHITECTURE

We have investigated several logic-based approaches for the domain and problem described above. The PDDL+ language has been designed to express planning problems in mixed discrete/continuous domains. As each discrete change is represented by a planning step, our test case problem instances would need at least 50-100 steps to be solvable. We were only able to solve the most trivial test cases in less than one second using the SMTPlan+ solver.

Encoding into SMT can be done by expressing planning as bounded model checking. This approach suffers from the same problem of having a high number of planning steps (however, some improvements can be made, such as making train driver choices implicit in constraints on the relation between velocity, distance and time).

As a consequence, we developed a CEGAR-style tool which exploits the limited number of control system commands to make an abstraction of the planning problem, see Fig. 4.

A verification tool chain which solves the low-level railway infrastructure capacity verification problem and supports agile
verification in railway construction projects is outlined in Fig. 5. The user manual and source code are online.

The tool is complementary to other verification techniques in railway design, such as static layout verification [17], [18], static interlocking verification [20], [18], interlocking program verification [21], and timetable analysis [11].

The following input documents are used:

- **Operational scenarios** defining the performance properties to verify. Examples are given in Section II-B.
- **Infrastructure** given in the railML format [22], [23]. In our case studies we used the RailCOMPLETE software, a plugin for the widely used AutoCAD drafting software. Using a model taken directly from the drafting program means that no additional model preparation is needed.
- **Elementary routes (optional)**, given in a custom format which is compatible with the upcoming railML interlocking format. Although subject to design, a decent guess of the content can be straight-forwardly derived from the infrastructure by listing resources in paths between adjacent signals, so this input is optional.
- **Dispatch plan (optional)**, which demonstrates adequate performance is needed. To avoid specifying a dispatch plan, the verification tool can produce a plan which fulfills the performance specifications.

An advantage of the separation of planner and simulator is that each component can be used separately. The planner alone may be used to enumerate different possibilities for train movements, which might be used in an operational testing situation. The simulator alone may be used to debug the execution of a specific dispatch plan to examine performance deficiencies, and educationally for demonstrating the workings of the railway system. Put together, the components provide automated verification, which is the main goal of our efforts.

It would also, in principle, be possible to use one of the commercial simulation packages, such as OpenTrack or RailSys, provided that all input and simulation control can be provided though a programmable interface (API).

### A. Timing Evaluation using Simulation

Given a specific dispatch plan, we evaluate the time needed for executing it using discrete event simulation (DES), where a set of concurrent processes operate on a shared system state. Processes execute by reading or writing to the shared state, firing events, and then going to sleep until a specific event fires or a given amount of time has passed. When all processes are sleeping, the simulation timer is advanced to the earliest time when a process is scheduled to wake up.

Our DES for railway simulation has the following processes:

1) **Elementary route activation**: waits for resources, allocates them, sets switches to given positions and starts the following sub-processes:

   - **Release trigger**: listens to a trigger detection section which is designated as the release trigger for a partial route. After the detection section has first been occupied, and later freed, resources are released for use in other elementary routes.
   - **Signal catcher**: sets the route entry signal to the 'proceed' aspect, then waits for a given trigger section to become occupied before setting the signal to back 'stop'.

2) **Train**: evaluates movement authority using information from signals currently in sight, and takes one of the following actions: accelerate, brake, or coast/wait. Braking curves from velocity limitations are calculated, representing the train driver's plan for when to start braking. A guaranteed minimum time until further action is required from the driver is calculated by taking the minimum time until one of the following happen:
Braking the transition relation of the system for a number of steps \( k \), we use bounded model checking (BMC) to unroll the transition relation of the system into an instance of the Boolean satisfiability problem (SAT).

We consider the problem a model checking problem, and use BMC to encode an instance of the abstracted planning problem of visiting targets. The SAT instance is built incrementally by solving with \( k \) steps and then adding the \( k^{th} \) step if necessary.

The plan fulfills performance specifications:

- Each route \( r_j \) has an occupancy status \( o^i_{r_j} \): it can be free \( (o^i_{r_j} = \text{Free}) \) or it can be occupied by a specific train \( t_k (o^i_{r_j} = t_k) \). Each combination of route and train is represented by a Boolean variable, but we will write constraints with \( o^i_{r_j} \) as a variable from the set of trains.
- Each train \( t_k \) has a Boolean representing appearance status \( \delta^i_{t_k} \), used to propagate to future states that a train has started (used in constraint C2).
- Each visit \( l \) has a Boolean representing required visits \( v^j_l \), which is used to propagate to future states that a visitation requirement has been fulfilled (used in constraint C5).
- Each combination of route \( r_j \) and train \( t_k \) has a Boolean representing deferred progress \( p^i_{r_j,t_k} \), used to propagate to future states that a train is not progressing, and must resolve the conflict in the future (used in constraint C8).

A dispatch plan is produced directly from the occupancy status \( o^i_{r_j} \) of states by taking the difference between consecutive states and then dispatching any trains and routes which become active from one state to the next.

Constraints are applied to each state and each pair of consecutive states to ensure that:

- The plan is viable for execution (i.e., correctness):
  - (C1) Conflicting routes are not activated simultaneously.
  - (C2) Each train can only take one continuous path.
  - (C3) Elementary routes are allocated as a unit, but its parts may be deallocated separately.
  - (C4) (Partial) routes are deallocated only after a train has fully passed over them.
- The plan fulfills performance specifications:
  - (C5) Trains perform their specified visits.
  - (C6) Visits happen in specified order.
- Equivalent solutions are eliminated (for performance):
  - (C7) Routes are deallocated immediately after the train has fully passed over them.
  - (C8) A train’s path is extended as far as possible in the current time step, unless hindered by a conflicting train.

Equivalent plans, which result in the same trains traversing the same paths and conflicting in the same locations, should have the same representation so that enumeration of different
plans produces meaningful alternatives. This equivalence is demonstrated in the crossing example in Fig. 1 where the two alternative plans are the only alternatives given by the planner. The simulator component will rule out prefixes of plans, which are then eliminated from the SAT problem by adding constraints incrementally.

The implementation of each of these constraints as propositional logic statements is described below. Constraints apply separately to all states \( i \) unless noted otherwise.

1) Resource conflicts (C1): Any two routes which require the same resources cannot both be allocated in the same state.

\[
\forall r_a \in \text{Routes} : \forall r_b \in \text{conflict}(r_a) : o^i_{r_a} = \text{Free} \lor o^i_{r_b} = \text{Free}.
\]

2) Train path (C2): At most one alternative route is taken by a train in a single state. First, ensure that only one route from a given start signal may be taken at any time.

\[
\forall t \in \text{Trains} : \forall s \in \text{Signal} : \text{atMostOne}\left(\{ o^i_t = t \mid \text{entry}(r) = s \}\right)
\]

Note that entry signals for all routes entering from a model boundary share the same null value, so that this constraint also excludes plans where a single train appears in several positions at once. Each train should only enter the plan once, thus the appearance Boolean changes to true in exactly one transition.

\[
\forall t \in \text{Trains} : b^i_t \Rightarrow b^{i+1}_t.
\]

\[
\forall t \in \text{Trains} : \text{exactlyOne}\left(\left\{ -b^j_t \land b^{j+1}_t \mid j \in \text{States} \right\}\right).
\]

When an entry boundary route is allocated, it must be the train’s appearing state:

\[
\forall t \in \text{Trains} : \forall r \in \{ r \in \text{Routes} \mid \text{entry}(r) = \text{null} \} : (o^i_t \neq t \land o^{i+1}_r = t) \Rightarrow b^{i+1}_t.
\]

Routes which are not entry routes can only be allocated to a train when they extend some other route which was already allocated to the same train, i.e. consecutive routes must match so that the exit signal of one is the entry signal of the next:

\[
\forall t \in \text{Trains} : \forall r \in \{ r \in \text{Routes} \mid \text{entry}(r) \neq \text{null} \} : (o^i_t \neq t \land o^{i+1}_r = t) \Rightarrow \bigvee \{ o^{i+1}_r \mid r \in \text{Routes}, \text{where entry}(r) = \text{exit}(r) \}
\]

3) Partial release (C3): Partial release is represented by splitting each elementary route into separate routes for each component which is released separately. The set Partial contains such sets of routes. Partial routes are allocated together:

\[
\forall t \in \text{Trains} : \forall q \in \text{Partial} : \text{allEqual}\left(\{ o^i_t \neq t \land o^{i+1}_r = t \mid r \in q \}\right)
\]

4) Deallocation (C4, C7): Routes are freed when sufficient routes have been allocated ahead to fully contain the length of the train.

\[
\forall t \in \text{Trains} : \forall r \in \text{Routes} : o^i_t = t \Rightarrow (o^{i+1}_r = t) = \text{freeable}_{r,t}(\{ o_i \})
\]

Note that the equality sign on the right hand side implies that deallocation is both allowed (C4), and required (C7). The freeable predicate is a disjunction of paths (conjunction of routes) ahead which are long enough to contain the train.

5) Visits (C5, C6): Visits and their order are given by the set \( V_i \), which contains pairs of \( (t, v) \), where \( t \) is a train and \( v \) is a set of alternative routes. Visits must happen using any of the alternative routes, and must be in an order such that the visit \((t_1, v_1)\) comes before \((t_2, v_2)\):

\[
\forall r(a(t_1, v_1), (t_2, v_2)) \in V_i : \bigvee \{ o^i_{r_a} = t_1 \land o^i_{r_b} = t_2 \land i \leq j \}
\]

When \( i \) is the last state, \( p^1 \) is considered to be false, which forces the deferred progress to be resolved eventually. Note that it is not required that the conflicting trains are distinct.

IV. CASE STUDIES AND PERFORMANCE

This section presents running times for different typical performance specifications on different types of railway infrastructure where the size and complexity of the model is typical for the scope of railway construction projects. Verification performance on various test examples as well as real stations is presented in Table 1. The table shows the calculation time required in each solver component, and also shows the number of invocations \( n_{DES} \) of the simulator, which is very low in most of the practical cases. This supports our hypothesis that the chosen abstraction and CEGAR loop is efficient.

The two-track station used in Fig. 1 is not too complex, having 6 signals, 2 switches, and 6 routes. Even so, this scale is still interesting for verification in practice, since there are many possible mistakes to uncover, and also because it runs so fast that it is possible to run it after each design change.

The Norwegian railway infrastructure manager Bane NOR has supplied a railML infrastructure model of the whole national railway network [23] from which we have extracted some more complex examples. Fig. 8 shows cut-outs from the visual representation of these models, i.e., the stations Kolbotn, Eidsvoll, and Asker were converted from the railML models.
We have also tested against an infrastructure model from the Arna construction project that uses the RailCOMPLETE CAD design software, a realistic use case for agile verification.

Finally, to test the limitations of scalability in our method, we construct a set of examples where \( m \) stations each with \( n \) parallel tracks each are serially connected by a single track. In this case, when a timing bound is slightly too small to be satisfiable, the planner will have to come up with \( n^m \) plans for timing evaluation. This scenario is outside the intended use case for our method: path selection can on this scale instead be based on static speed profiles. Capacity over many stations is better suited for the established timetabling tooling.

We attempted an alternative implementation using the PDDL+ solver SMTPlan+, but found that even for greatly simplified models, the required number of steps and numerical constraints put all our case studies out of reach for sub-second verification times.

V. RELATED WORK

Railway timetabling and capacity analysis has often been posed as a planning problem and solved using mixed integer programming and similar approaches. Zwaneveld et al.\(^ {25} \) use integer programming on a problem closely related to our low-level railway infrastructure capacity verification problem. Isobe et al.\(^ {26} \) also formulate a similar model in timed CSP, representing train locations and velocities, and control system logic. Our definition of the problem in this paper includes non-linear constraints on train movement dynamics (maximum acceleration/braking power) and communication constraints (trains must slow down if they have not been informed of movement authority), which are relevant in construction projects but less relevant in timetabling.

Many variations on discrete event simulation are used in railway dynamic analysis, see e.g.\(^ {27} \),\(^ {28} \),\(^ {29} \).

In the planning literature, the PDDL+ language\(^ {3} \) has been introduced to capture mixed discrete/continuous planning problems such as the one studied in this paper. General-purpose solvers have recently been developed, using time domain discretization (DiNo\(^ {30} \)) or the SMT theory of non-linear real arithmetic (SMTPlan+\(^ {31} \)).

VI. CONCLUSIONS AND FURTHER WORK

The goal of our suggested tool chain for railway engineering is (1) to allow fully automated performance properties verification and (2) use minimal input documentation for the verification. Both of these aspects encourage bringing in performance verification into frequently changing early-stage design projects, avoiding the costly and time-consuming backtracking required when later-stage analysis reveals unacceptable performance.

As future work we plan to integrate the current prototype in the RailCOMPLETE tool and test the usability with the engineers using this tool in design work.

Acknowledgments: We thank the engineers at Railcomplete AS, especially senior engineer Claus Feyling, for guidance on railway operations and design methodology.
REFERENCES


