Design-Time Railway Capacity Verification using SAT modulo Discrete Event Simulation

Bjørnar Luteberget
Railcomplete AS
Sandvika, Norway
Email: bjornar.luteberget@railcomplete.no

Koen Claessen
Chalmers University of Technology
Gothenburg, Sweden
Email: koen@chalmers.se

Christian Johansen
University of Oslo
Oslo, Norway
Email: cristi@ifi.uio.no

***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. Verifying these properties amounts to solving a planning problem constrained by discrete control system logic, network topology, laws of motion, and sparse communication. To describe train dynamics one uses second-order linear differential equations which when solved analytically give rise to non-linear equations over real variables.

We argue that reasoning over the whole discrete/continuous solution space is not efficient with current state-of-the-art solvers. Instead, we have solved the problem by building a special-purpose solver which splits the problem into two: an abstracted SAT-based dispatch planning, and continuous-domain dynamics and timing constraints evaluated using discrete event simulation. 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: Does the station infrastructure have the *capacity* to handle the amount of trains and the desired traveling times to provide adequate service in transportation of goods and passengers?

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.

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

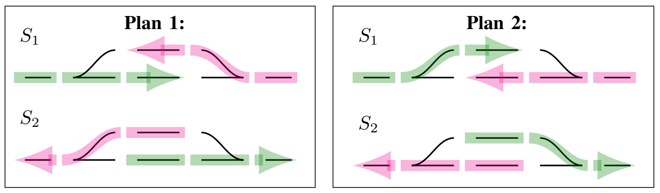


Fig. 1: Two alternative plans for achieving a crossing of two trains on a two-track station. The green areas show track segments which are currently occupied by a train going from left to right, while the pink areas show track segments which are currently occupied by a train going from right to left.

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 a verification technique and tool to help engineers specify capacity properties *at design time* 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 consider **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], [3] or PDDL+ [4] languages, essentially resulting in a SAT modulo non-linear real arithmetic problem [5], [6]. Many solvers can handle such problems [7], [8], [9], 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 simple 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) [10] 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) [11] 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: Sec. 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. Sec. 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. Sec. IV contains performance evaluations in a set of relevant case studies. Sec. V gives pointers to related work, and Sec. VI presents our conclusions.

II. Domain background and problem description

Railway capacity is hard to define precisely (see [12], [13] 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 [14], OpenTrack [15], or LUKS [16]. Good overviews of methods are presented in [17] and [18].

The so-called analytical approaches to capacity analysis using networked queuing theory [19], maximum flow (originally posed as a railway capacity problem [20]), or max-plus algebra [21], 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.

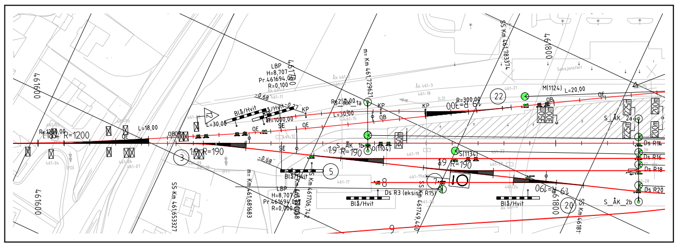
Railway design

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 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)

Elementary route	Start signal	End signal	Switch position	Track segments	Conflicts
AC	A	C	X right	1, 2, 4	AE, BF
AE	A	E	X left	1, 2, 3	AC, BD
BF	B	F	Y left	4, 5, 6	AC, BD
BD	B	D	Y right	3, 5, 6	AE, BF

(b)

Fig. 2: Railway design artifacts: (a) Cut-out from 2D geographical CAD model (construction drawing) of preliminary design of the Arna station signalling. (b) Simplified example of tabular interlocking (control system) specifications.

correctness constraints (described in Sec. II-A below), demonstrate the properties described in the performance requirements (detailed in Sec. II-B below).

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 from the dispatcher for activating **elementary routes**. An elementary route is the 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.

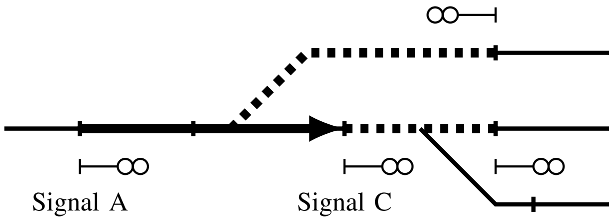


Fig. 3: Elementary route AC from signal A to the adjacent signal C. The thick line indicates track segments on the train's path which are reserved for this movement, and the dashed lines indicate reserved track segments outside the path.

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 Δt 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 [22] 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 $v_{\max}$, 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 $\{l_i\}$ 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 visits are only required to be ordered, so that $t_{q_a} < t_{q_b}$.

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:

```
vehicle passengertrain length 220.0
  accel 1.0 brake 0.9 maxspeed 55.0
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.

```
movement passengertrain {
  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.

```
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.

```
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.

```
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 BMC. This approach suffers from the same problem of having a high number of planning steps (some improvements can be made, s.a. making train driver choices implicit in constraints on the relation between velocity, distance and time).

In response to all these, 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

[1]For details of the input file formats, see https://luteberget.github.io/rollingdocs/usage.html

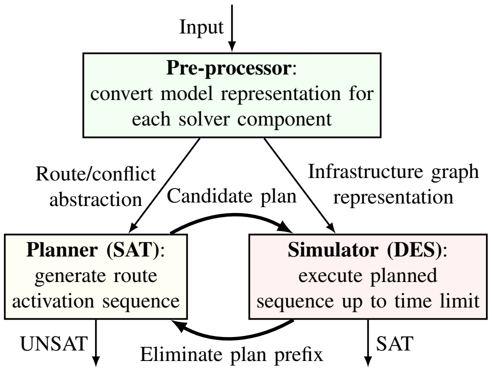

Input
Pre-processor: convert model representation for each solver component
Route/conflict abstraction
Infrastructure graph representation
Candidate plan
Planner (SAT): generate route activation sequence
Simulator (DES): execute planned sequence up to time limit
UNSAT
Eliminate plan prefix
SAT

Fig. 4: Conceptual diagram of CEGAR architecture. Infrastructure, routes, train types, and movement specifications are transformed into (1) the planner's abstract representation, containing only elementary routes and train lengths, and (2) the detailed graph representation used in the simulator component.

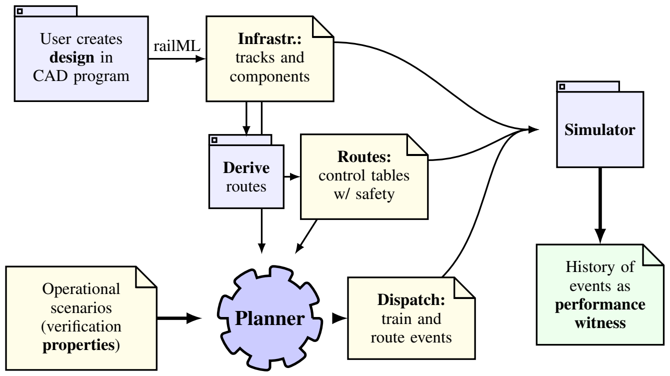

User creates **design** in CAD program
railML
Infrastr.: tracks and components
Simulator
Derive routes
Routes: control tables w/ safety
History of events as **performance witness**
Operational scenarios (verification **properties**)
Planner
Dispatch: train and route events

Fig. 5: Verification tool chain overview. Yellow boxes represent input documents. Note that only infrastructure and operational scenarios are strictly required – interlocking tables can be derived, and dispatch plans can be synthesized. Blue boxes represent programs. The green box represents the output document from the simulator, which is a history of events which is the witness that proves the performance requirement.

verification in railway construction projects is outlined in Fig. 5. The manual, source code and test cases are available online[2]. The tool uses the MiniSAT v2.2.0 solver.

The tool is complementary to other verification techniques in railway design, such as static layout verification [23], [24], [25], static interlocking verification [26], [24], interlocking program verification [27], and timetable analysis [17].

The following input documents are used:

- **Operational scenarios** defining the performance properties to verify. Examples are given in Sec. II-B.
- **Infrastructure** given in the railML format [28], [29]. 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 plans** (*optional*) corresponding to each operational scenario. The verification tool can produce dispatch plans fulfilling the performance specification, so this input is optional.

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.

[2]https://luteberget.github.io/rollingdocs and https://github.com/koengit/trainspotting

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 given 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 (corr. Sec. II-A2): 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 (corr. Sec. II-A3 and Sec. II-A4): 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 (see also Fig. 6):

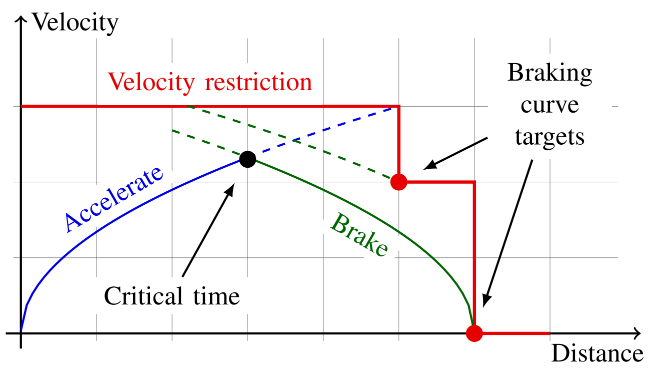


Fig. 6: The train driver's decision about when to accelerate/brake/coast happens at intersections between acceleration curves, braking curves and velocity restriction curves. In this example, the train can accelerate until the critical time where the acceleration intersects with the braking curve towards the second velocity restriction ahead (the first one is not critical).

- train arrives at a new node
- train reaches maximum velocity
- train enters the area of a new velocity restriction
- acceleration/coasting curve intersects braking curve

After this minimum time has passed, or any signals currently in sight have changed state, the train updates its position and velocity according to the chosen driver action and the laws of motion. Note that since we assume a constant maximum acceleration and braking, the equations of motion can be solved analytically, and there is no need for discretizing the time or space domains, except for the re-evaluation of the equations of motion at discrete events. This ensures that the train starts braking in time using only the information available to the driver at any given time.

B. *Dispatch Planning using SAT*

The planner solves the abstracted discrete planning problem of finding a dispatch plan, i.e. determining a sequence of trains and elementary routes which make the trains end up visiting locations according to the movements specification.

We encode an instance of the abstracted planning problem into an instance of the Boolean satisfiability problem (SAT). We consider the problem a model checking problem, and use the technique of bounded model checking (BMC) to unroll the transition relation of the system for a number of steps k, expressing state and transitions using propositional logic.

Using BMC for planning works by asserting the existence of a plan, so that when the corresponding SAT instance is satisfiable, it proves the fulfillment of the performance requirements and gives an example plan for it. When unsatisfiable, we are ensured that there is no plan within the number of steps k. In practice plans with higher number of steps are not of interest; i.e., the bound k is chosen based on practical considerations (twice the number of trains was sufficient in our case study). The SAT instance is built incrementally by solving with $k-1$ steps and then adding the k^{th} step if necessary.

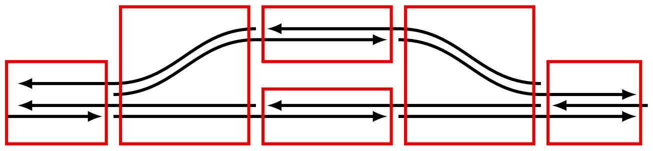


Fig. 7: The planner component takes an abstracted view of the railway infrastructure. Lines represent elementary routes with traveling direction given by the arrows. Boxes indicate routes in conflict, i.e. only one of them can be in use at a time.

The abstracted planning problem is encoded as a SAT instance by representing states, constraints on each state, and constraints on consecutive states. *State* i of the system in the planner component is represented as:

- 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 ($o^j_{r_i} = t_k$). 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** b^i_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^i_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** $p^i_{j,k}$, 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 on states ensure the following:

- 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) An elementary route must be 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 plans shown are the *only* alternatives given by the planner.

The simulator component, which evaluates the time consumption of plans, reports which parts of the plan fail the timing constraints, and the negation of this partial plan is added to the SAT instance. Since the timing calculations are path dependent, we use the part of the plan starting from the beginning and going up to the step where the timing specification violation occurs. This way of refining the abstraction can cause performance problems when many different choices are possible early in the plan, and the timing violation can only be found near the end of the plan, as demonstrated in Sec. IV. Finding a way to make more precise refinements could be necessary for larger problem instances.

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.

$$\forall t \in \text{Trains} : \forall s \in \text{Signal} :$$
$$\text{atMostOne}(\{o^i_r = t \mid \text{entry}(r) = s\})$$

We use a standard sequential encoding to encode atMostOne and other similar constraints, as explained in e.g. [30]. 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.

$$\forall t \in \text{Trains} : b^i_t \Rightarrow b^{i+1}_t.$$
$$\forall t \in \text{Trains} : \text{exactlyOne}\left(\left\{\neg b^j_t \land b^{j+1}_t \mid j \in \text{States}\right\}\right),$$

A train appears when an entry boundary route is allocated:

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

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:

$$\forall t \in \text{Trains} : \forall r \in \{r \in \text{Routes} \mid \text{entry}(r) \neq \text{null}\} :$$
$$\left(o^i_r \neq t \land o^{i+1}_r = t\right) \Rightarrow$$
$$\bigvee \{o^{i+1}_{r_x} = t \mid r_x \in \text{Routes}, \text{entry}(r) = \text{exit}(r_x)\}$$

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:

$$\forall t \in \text{Trains} : \forall q \in \text{Partial} :$$
$$\text{allEqual}(\{o^i_r \neq t \land o^{i+1}_r = t \mid r \in q\})$$

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

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

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 VisitOrder, 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):

$$\forall((t_1, v_1), (t_2, v_2)) \in \text{VisitOrder} :$$
$$\bigvee \{o^i_{r_a} = t_1 \land o^j_{r_b} = t_2 \land i \leq j$$
$$\mid r_a \in (v_1), r_b \in (v_2), i, j \in \text{States}\}$$

6) Forced progress (C8): In addition to the constraints on allocation and freeing that are required to produce a valid plan, we also add constraints which force each train to get allocated routes further along a path forward unless there is a conflict. Routes ahead are either allocated, or the train is deferred p:

$$\forall t \in \text{Trains} : \forall r \in \text{Routes} :$$
$$o^i_r \Rightarrow p^i_{t,r} \lor \bigvee \{o^i_{r_x} \mid r_x \in \text{Routes}, \text{entry}(r_x) = \text{exit}(r)\}$$

Deferred progress must be resolved by freeing a conflicting route, and then allocating it to the train in the following step:

$$\forall t \in \text{Trains} : \forall r \in \text{Routes} :$$
$$p^i_{t,r} \Rightarrow p^{i+1}_{t,r} \lor \bigvee \{o^i_{r_c} \neq \text{Free} \land o^i_{r_x} \neq t \land o^{i+1}_{r_x} = t$$
$$\mid r_c, r_x \in \text{Routes}, \text{exit}(r) = \text{entry}(r_x), r_c \in \text{conflict}(r)\}$$

When i is the last state, $p^{i+1}_{t,r}$ 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 I. The table shows the time spent 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 only 6 elementary routes. Even so, this scale is still interesting for verification in practice, since there are many possible mistakes to uncover.

The Norwegian railway infrastructure manager Bane NOR has supplied a railML infrastructure model of the whole national railway network [31] from which we have extracted some more complex examples. Fig. 8 shows cut-outs from the

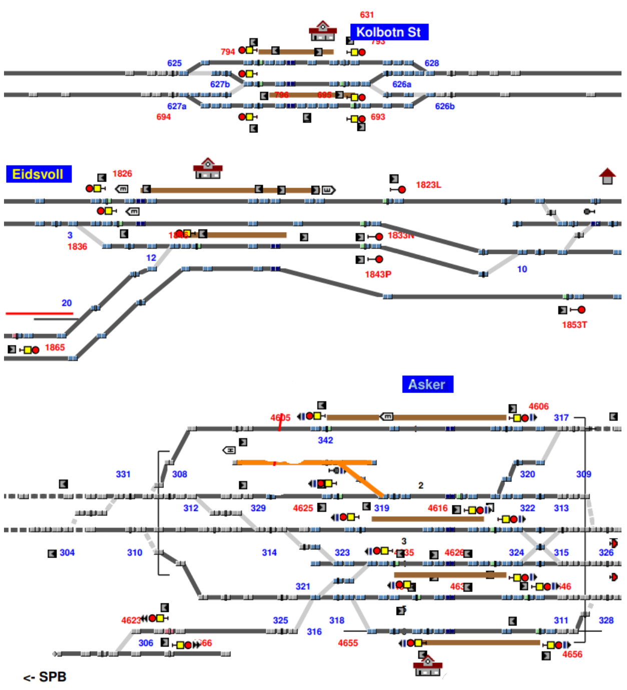


Fig. 8: Stations Kolbotn, Eidsvoll, and Asker from Bane NOR's model of the Norwegian national network [31].

Infrastructure	Property	Result	n_{DES}	t_{SAT}	t_{DES}	t_{total}
Simple (3 elem.)	Run.time	Sat.	1	0.00	0.00	0.00
	Crossing	Unsat.	0	0.00	0.00	0.00
Two track (14 elem.)	Run.time	Sat.	1	0.01	0.00	0.01
	Frequency	Sat.	1	0.01	0.00	0.01
	Overtaking 2	Sat.	1	0.00	0.00	0.01
	Overtaking 3	Unsat.	0	0.01	0.00	0.01
	Crossing 3	Unsat.	0	0.01	0.00	0.01
Kolbotn (BN) (56 elem.)	Run. time	Sat.	2	0.01	0.00	0.02
	Overtake 4	Sat.	1	0.05	0.00	0.06
	Overtake 3	Unsat.	0	0.05	0.00	0.06
Eidsvoll (BN) (64 elem.)	Run. time	Sat.	2	0.01	0.00	0.02
	Overtake 2	Sat.	1	0.08	0.00	0.08
	Crossing 3	Sat.	1	0.04	0.00	0.04
	Crossing 4	Unsat.	0	0.21	0.00	0.21
Asker (BN) (170 elem.)	Overtaking 2	Sat.	1	0.20	0.00	0.21
	Overtaking 3	Unsat.	1	0.73	0.00	0.74
	Crossing 4	Sat.	0	0.75	0.00	0.77
Arna (CAD) (258 elem.)	Run. time	Sat.	1	0.02	0.00	0.04
	Overtaking 2	Sat.	1	0.50	0.00	0.51
	Overtaking 3	Sat.	1	1.43	0.00	1.45
	Crossing 4	Sat.	1	1.73	0.00	1.74
Gen. 3x3 (74 elem.)	High time	Sat.	1	0.01	0.00	0.01
	Low time	Unsat.	27	0.18	0.01	0.19
Gen. 4x4 (196 elem.)	High time	Sat.	1	0.01	0.00	0.03
	Low time	Unsat.	256	2.08	0.26	2.34
Gen. 5x5 (437 elem.)	High time	Sat.	1	0.06	0.00	0.09
	Low time	Unsat.	3125	38.89	4.35	43.24

TABLE I: Verification performance on test cases, including Bane NOR (BN) and RailCOMPLETE (CAD) infrastructure models. The number of elementary routes (*elem.*) is shown for each infrastructure to indicate the model's size. n_{DES} is the number simulator runs, t_{SAT} the time in seconds spent in SAT solver, t_{DES} the time in seconds spent in DES, and t_{total} the total calculation time in seconds.

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. [32] use integer programming on a problem closely related to our low-level railway infrastructure capacity verification problem. Isobe et al. [33] formulate a similar model in timed CSP, representing train locations, velocities, and control logic. Our definition of the problem in this paper includes non-linear constraints on train dynamics (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. [34], [35], [36].

In the planning literature, the PDDL+ language [4] 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 [37]) or the SMT theory of non-linear real arithmetic (SMTPlan+ [38]).

VI. Conclusions and Further Work

The goal of our suggested tool chain for railway engineering is (1) to allow fully automated performance 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 their design work.

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

References

[1] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*. Springer, 2018, pp. 305–343. [Online]. Available: https://doi.org/10.1007/978-3-319-10575-8_11

[2] L. De Moura and N. Bjørner, "Satisfiability modulo theories: introduction and applications," *Communications of the ACM*, vol. 54, no. 9, pp. 69–77, 2011. [Online]. Available: https://doi.org/10.1145/1995376.1995394

[3] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T)," *J. ACM*, vol. 53, no. 6, pp. 937–977, 2006. [Online]. Available: http://doi.acm.org/10.1145/1217856.1217859

[4] M. Fox and D. Long, "Modelling mixed discrete-continuous domains for planning," *J. Artif. Intell. Res.*, vol. 27, pp. 235–297, 2006. [Online]. Available: https://doi.org/10.1613/jair.2044

[5] M. Franzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, "Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 1, pp. 209–236, 2007. [Online]. Available: https://satassociation.org/jsat/index.php/jsat/article/view/16

[6] D. Jovanovic and L. de Moura, "Solving non-linear arithmetic," *ACM Comm. Computer Algebra*, vol. 46, no. 3/4, pp. 104–105, 2012. [Online]. Available: http://doi.acm.org/10.1145/2429135.2429155

[7] L. M. de Moura and N. Bjørner, "Z3: an efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, ser. Lecture Notes in Computer Science, C. R. Ramakrishnan and J. Rehof, Eds., vol. 4963. Springer, 2008, pp. 337–340. [Online]. Available: https://doi.org/10.1007/978-3-540-78800-3_24

[8] S. Gao, S. Kong, and E. M. Clarke, "dReal: An SMT solver for nonlinear theories over the reals," in *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction*, ser. Lecture Notes in Computer Science, M. P. Bonacina, Ed., vol. 7898. Springer, 2013, pp. 208–214. [Online]. Available: https://doi.org/10.1007/978-3-642-38574-2_14

[9] B. Dutertre, "Yices 2.2," in *Computer Aided Verification - 26th Conf., CAV 2014*, ser. Lecture Notes in Computer Science, A. Biere and R. Bloem, Eds., vol. 8559. Springer, 2014, pp. 737–744. [Online]. Available: https://doi.org/10.1007/978-3-319-08867-9_49

[10] S. Robinson, *Simulation: The Practice of Model Development and Use*. USA: John Wiley & Sons, Inc., 2004.

[11] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, Y. Zhu *et al.*, "Bounded model checking." *Advances in computers*, vol. 58, no. 11, pp. 117–148, 2003. [Online]. Available: https://doi.org/10.1016/S0065-2458(03)58003-2

[12] M. Abril, F. Barber, L. Ingolotti, M. Salido, P. Tormos, and A. Lova, "An assessment of railway capacity," *Transportation Research Part E: Logistics and Transportation Review*, vol. 44, no. 5, pp. 774 – 806, 2008. [Online]. Available: https://doi.org/10.1016/j.tre.2007.04.001

[13] A. Landex, "Methods to estimate railway capacity and passenger delays," Ph.D. dissertation, 2008. [Online]. Available: http://orbit.dtu.dk/en/publications/id(f5578206-74c3-4c94-ba0d-43f7da82bf95).html

[14] "RMCon RailSys," 2018. [Online]. Available: http://www.rmcon.de/railsys-en/

[15] "OpenTrack: Simulation of railway networks," 2018. [Online]. Available: http://www.opentrack.ch/

[16] "LUKS: Analysis of lines and junctions," 2018. [Online]. Available: http://www.via-con.de/development/luks

[17] S. S. Harrod, "A tutorial on fundamental model structures for railway timetable optimization," *Surveys in Operations Research and Management Science*, vol. 17, no. 2, pp. 85 – 96, 2012. [Online]. Available: https://doi.org/10.1016/j.sorms.2012.08.002

[18] A. A. Assad, "Models for rail transportation," *Transportation Research Part A: General*, vol. 14, no. 3, pp. 205 – 220, 1980. [Online]. Available: https://doi.org/10.1016/0191-2607(80)90017-5

[19] L. Kleinrock, *Theory, Volume 1, Queueing Systems*. New York, NY, USA: Wiley-Interscience, 1975.

[20] T. Harris and F. Ross, "Fundamentals of a method for evaluating rail net capacities," Rand Corp., Tech. Rep., 1955.

[21] R. M. Goverde, "Railway timetable stability analysis using max-plus system theory," *Transportation Research Part B: Methodological*, vol. 41, no. 2, pp. 179 – 201, 2007, advanced Modelling of Train Operations in Stations and Networks. [Online]. Available: https://doi.org/10.1016/j.trb.2006.02.003

[22] J. Pachl, *Railway Operation and Control*. VTD Rail Publishing, 2015.

[23] B. Luteberget, C. Johansen, and M. Steffen, "Rule-based consistency checking of railway infrastructure designs," in *Integrated Formal Methods 2016*, ser. Lecture Notes in Computer Science, E. Ábrahám and M. Huisman, Eds., vol. 9681. Springer, 2016, pp. 491–507. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-33693-0_31

[24] B. Luteberget and C. Johansen, "Efficient verification of railway infrastructure designs against standard regulations," *Formal Methods in System Design*, vol. 52, no. 1, pp. 1–32, Feb 2018. [Online]. Available: https://doi.org/10.1007/s10703-017-0281-z

[25] B. Luteberget, J. J. Camilleri, C. Johansen, and G. Schneider, "Participatory Verification of Railway Infrastructure by Representing Regulations in RailCNL," in *International Conference on Software Engineering and Formal Methods (SEFM)*, ser. Lecture Notes in Computer Science, A. Cimatti and M. Sirjani, Eds., vol. 10469. Springer International Publishing, 2017, pp. 87–103. [Online]. Available: https://doi.org/10.1007/978-3-319-66197-1_6

[26] A. E. Haxthausen and P. H. Østergaard, "On the Use of Static Checking in the Verification of Interlocking Systems," in *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications*, T. Margaria and B. Steffen, Eds. Springer, 2016, pp. 266–278. [Online]. Available: https://doi.org/10.1007/978-3-319-47169-3_19

[27] A. Borälv and G. Stålmarck, "Formal verification in railways," in *Industrial-Strength Formal Methods in Practice*, M. G. Hinchey and J. P. Bowen, Eds. Springer, 1999, pp. 329–350. [Online]. Available: https://doi.org/10.1007/978-1-4471-0523-7_15

[28] A. Nash, D. Huerlimann, J. Schütte, and V. P. Krauss, "RailML — a standard data interface for railroad applications," in *Computers in Railways IX*. WIT Press, 2004, pp. 233–240.

[29] "railML. The XML interface for railway applications," 2018. [Online]. Available: http://www.railml.org

[30] C. Sinz, "Towards an optimal CNF encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming - CP 2005*, ser. Lecture Notes in Computer Science, P. van Beek, Ed., vol. 3709. Springer, 2005, pp. 827–831. [Online]. Available: https://doi.org/10.1007/11564751_73

[31] "Bane NOR: Model of the Norwegian rail network," 2016. [Online]. Available: http://www.banenor.no/en/startpage1/Market1/Model-of-the-national-rail-network/

[32] P. J. Zwaneveld, L. G. Kroon, and S. P. van Hoesel, "Routing trains through a railway station based on a node packing model," *European Journal of Operational Research*, vol. 128, no. 1, pp. 14 – 33, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221700000874

[33] Y. Isobe, F. Moller, H. N. Nguyen, and M. Roggenbach, "Safety and line capacity in railways – an approach in timed csp," in *Integrated Formal Methods*, J. Derrick, S. Gnesi, D. Latella, and H. Treharne, Eds. Springer, 2012, pp. 54–68. [Online]. Available: https://doi.org/10.1007/978-3-642-30729-4_5

[34] M. Montigel, "Modellierung und gewährleistung von abhängigkeiten in eisenbahnsicherungsanlagen," Ph.D. dissertation, ETH Zurich, 1994. [Online]. Available: https://www.research-collection.ethz.ch/handle/20.500.11850/47983

[35] D. Hürlimann, "Objektorientierte modellierung von infrastrukturelementen und betriebsvorgängen im eisenbahnwesen," Ph.D. dissertation, ETH Zurich, 2002. [Online]. Available: https://www.research-collection.ethz.ch/handle/20.500.11850/47957

[36] E. Kamburjan and R. Hähnle, "Uniform modeling of railway operations," in *Formal Techniques for Safety-Critical Systems FTSCS 2016*, ser. Communications in Computer and Information Science, vol. 694. Springer, 2016, pp. 55–71. [Online]. Available: https://doi.org/10.1007/978-3-319-53946-1_4

[37] W. M. Piotrowski, M. Fox, D. Long, D. Magazzeni, and F. Mercorio, "Heuristic planning for PDDL+ domains," in *International Joint Conference on Artificial Intelligence, IJCAI 2016*, S. Kambhampati, Ed. IJCAI/AAAI Press, 2016, pp. 3213–3219. [Online]. Available: http://www.ijcai.org/Abstract/16/455

[38] M. Cashmore, M. Fox, D. Long, and D. Magazzeni, "A compilation of the full PDDL+ language into SMT," in *International Conference on Automated Planning and Scheduling, ICAPS 2016*, A. J. Coles, A. Coles, S. Edelkamp, D. Magazzeni, and S. Sanner, Eds. AAAI Press, 2016, pp. 79–87. [Online]. Available: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13101