Safe and Time-Optimal Control for Railway Games

Shyam Lal Karra, Kim Guldstrand Larsen, Florian Lorber, and Jiří Srba

Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark

Abstract. Railway scheduling is a complex and safety critical problem that has recently attracted attention in the formal verification community. We provide a formal model of railway scheduling as a stochastic timed game and using the tool Uppaal Stratego, we synthesise the most permissive control strategy for operating the lights and points at the railway scenario such that we guarantee system's safety (avoidance of train collisions). Among all such safe strategies, we then select (with the help of reinforcement learning) a concrete strategy that minimizes the time needed to move all trains to their target locations. This optimizes the speed and capacity of a railway system and advances the current state-of-the-art where the optimality criteria were not considered yet. We successfully demonstrate our approach on the models of two Danish railway stations, and discuss the applicability and scalability of our approach.

1 Introduction

Railway networks are complex safety critical systems where one has to guarantee safety despite the unpredictable behaviour of external factors influencing the system operation. This unpredictable behaviour arises from the fact that the durations taken by trains to change positions in the railway network are influenced by human operators as well as other factors like weather conditions etc. In addition to that, trains can move concurrently on multiple independent tracks and it becomes hard to manually control the lights and points (switches) in order to avoid trains collisions or derailment. This becomes particularly important, once we try to increase the throughput in the railway network and minimize the train travel times, as this requires more concurrency where dangerous situations can be easily overlooked by a human operator. There is hence a clear demand on the employment of automatic methods that will assist with a safe and time-optimal operation of a railway network, and this is the main focus of our paper.

We shall first introduce our railway scheduling problem by an example. An instance of the problem is given in Figure 1 and consists of two *trains*, each travelling in a given (and fixed) direction. At any moment, each train is placed on a clearly identified part of a track called *section*. In our example, the train t_0 is located at the section s_1 and travels from the left end-point of the section to its right end-point, assuming that the duration of such a move has some predefined

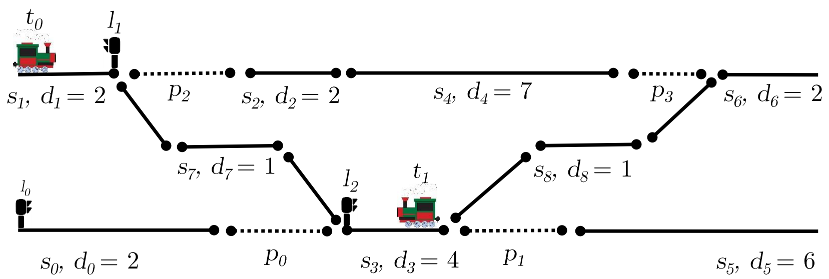


Fig. 1. A railway network with two trains t_0 and t_1, four points p_0, p_1, p_2 and p_3, nine sections $s_0 \ldots s_8$ and three lights l_0, l_1 and l_2.

probabilistic distribution with the expected value of 2 time units. Similarly, the train t_1 starts at the right end-point of section s_3 and moves in the right-to-left direction with the expected travel time of 4 time units to cross section s_3. Sections are connected with *points* (switches), like the point p_0 that depending on its *mode* can connect the section s_3 with either the section s_7 as depicted in our figure, or with the section s_0. Ends of sections can be guarded by *lights* that signal to the trains whether they are allowed to leave a given section and move to the connected section (depending on the mode of the points). We assume that passing a point is instantenious and that trains follow the light signals (i.e. never pass a light that is red).

The *railway scheduling problem* (also known as dynamic routing problem) is given by a track layout together with the initial positions of trains in the network and the target positions (sections) for each train. The role of the controller is to operate the lights and points such that the trains move to their target positions in a safe way (without the possibility of a collision, under the assumption that they respect the lights), while at the same time optimizing the time it takes before all trains arrive to their target sections.

Returning to our example, let us assume that the target section for the train t_0 is s_6 and that the trains t_1 aims to reach s_0. Note that we can change the points so that both trains simply drive straightforward and reach the target sections without the risk of a crash. However, the controller has an alternative choice for the train t_0 to navigate it through the sections s_7, s_3 and s_8 instead, where the expected arrival time is shorter than passing through the section s_4. Now, the optimal strategy depends on the fact how fast the train t_1 leaves the section s_3. If it does so early (depending on the respective stochastic distribution), then it is worth for the controller to direct the train t_0 along the faster route that includes the section s_3, otherwise (in case t_1 is for some reason delayed) it is more time-optimal to guide the train t_0 in the straight line.

In our work, we propose a method to construct such smart/adaptive controllers that guarantee both safety and time optimal behaviour. We propose a two-step controller synthesis procedure, which in the first step ensures safety, and in the second step optimises the controller to reduce the time needed for all trains to reach their goals. To achieve this, we first introduce a formal model of railway games and provide its encoding to the tool UPPAAL STRATEGO [6]. UPPAAL STRATEGO first computes the most permissive strategy for the given railway game represented as a two-player game. Here the controller decides the modes of points and light configurations, while the opponent (environment) moves the trains around in the network, respecting the lights guarding each section. The speed of the trains in this game is completely unconstrained, meaning that they may stay in a section forever or move immediately to the other end of the section without any time contraints. This guarantees that the synthetised strategy is safe, irrelevant of the speed of the trains (including the possibility that a train breaks and does not move at all). The tool UPPAAL STRATEGO is able to synthesise the most permissive strategy that includes *all* safe operations of the railway network. In the second step, we add stochastic behaviour to the trains so that they take a random amount of time for crossing a section which is chosen according to the rates for a given probabilistic distribution associated with the train and the concrete section in the network. From all safe strategies, we then (again using UPPAAL STRATEGO) select the fastest one by employing reinforcement learning. This provides us with the control strategy that is (near) time-optimal and provably safe.

Our main contributions can be summarized as follows:

– We generalize previous attempts to solve the train scheduling problem by allowing for concurrent moves of trains and we extend the existing railway models with stochastic information about the expected times for trains to travel along a section.
– We provide both the untimed and timed semantics for a railway network that allows us to argue about safety and time-optimality and we show their encoding in the model of stochastic timed automata.
– We explain the application of the tool UPPAAL STRATEGO for solving network scheduling problems and by means of two existing railway stations in Aalborg and Lyngy, we prove that the tool is able to compute safe and near-optimal control strategies.

Related work. Railway safety was studied e.g. in [14,16,3,5] with a focus on model checking of safety properties in railway networks, however, these approaches do not consider controller synthesis.

The work in [12] considers controller synthesis problem for the railway scheduling problem and the safety condition by translating it to strictly alternating two-player game and provides a number of abstractions in order to reduce the state space of the underlying game graph. In [10] this approach has been explored with the use of on-the-fly controller synthesis techniques, however, these approaches do not allow for concurrent train moves in order to reduce the complexity of the problem. The controller synthesis problem for railway network safety and deadlock avoidance was addressed in [9] by its modelling in Petri nets. In [11] automatic generation and verification of formal safety conditions from an interlocking tables of a relay interlocking system are discussed. However, none of these approaches consider the timing aspects in combination with controller synthesis and to the best of our knowledge, our work is the first one that synthetizes a schedule that is both safe and time-optimal at the same time.

Uppaal Stratego has been used in several case studies, where first a safe strategy is generated, and then an optimization step is performed. Examples include adaptive cruise control [13] and the intelligent control of traffic lights [8]. While the approach for the controller synthesis used in this paper is similar to these works, it is applied to a completely new area, which requires an efficient modeling of problem and nontrivial effort in order to boost the performance of the tool, including the use of a new learning method recently implemented in Uppaal Stratego.

2 Formal Definition of Railway Games

We first introduce a game graph that is used to give the semantics to our railway scheduling problem. A *game graph* is a tuple $\mathcal{G} = \langle V, \longrightarrow, \dashrightarrow, v_0, \mathit{Bad}, \mathit{Goal} \rangle$ where V is a finite set of vertices, $\longrightarrow \subseteq V \times V$ is the set of controller transitions, $\dashrightarrow \subseteq V \times V$ is the set of environmental transitions, $v_0 \in V$ is the initial vertex, $\mathit{Bad} \subseteq V$ is the set of bad vertices where controller loses and $\mathit{Goal} \subseteq V$ is the set of vertices controller aims to eventually reach.

Given a game graph $\mathcal{G}$, a *run* is a finite or infinite sequence $\rho = \langle v_0, v_1, v_2, \ldots \rangle$ of vertices such that either $v_i \longrightarrow v_{i+1}$ or $v_i \dashrightarrow v_{i+1}$ for every relevant i. A run ρ is a *maximal run* if it is either infinite or $\rho = \langle v_0 \ldots v_k \rangle$ and there is no v_{k+1} such that $v_k \longrightarrow v_{k+1}$ or $v_k \dashrightarrow v_{k+1}$. The set of all maximal runs starting at the vertex v_0 is denoted by $\mathit{MaxRuns}(v_0)$.

A controller *strategy* $\sigma : V \hookrightarrow V$ is a partial function such that for every $v \in V$ we have $v \longrightarrow \sigma(v)$ or $\sigma(v)$ is undefined in case that v has no outgoing controllable transitions. Given a strategy σ, the outcome of the game under σ from the vertex v_0 is defined as the set of all possible maximal runs that follow the strategy σ, formally

$$\mathit{Outcome}^{\sigma}(v_0) =$$
$$\{\langle v_0, v_1, \ldots \rangle \in \mathit{MaxRuns}(v_0) \mid v_i \dashrightarrow v_{i+1} \text{ or } \sigma(v_i) = v_{i+1} \text{ for all } i\} \ .$$

A run $\rho = \langle v_0, v_1, \ldots \rangle$ is called *safe* if there is no i such that $v_i \in \mathit{Bad}$. A strategy σ is a *safe strategy* if all the runs from $\mathit{Outcome}^{\sigma}(v_0)$ are safe. A strategy σ is a *winning strategy* if it is safe and for every run $\rho = \langle v_0, v_1 \ldots \rangle \in \mathit{Outcome}^{\sigma}(v_0)$ there is an i such that $v_i \in \mathit{Goal}$.

2.1 Railway Topology

We are now ready to formalise the railway topology. A *railway topology* is a tuple $R = (S, P, L, \gamma, E)$ where $S = \{s_0, s_1 \ldots s_m\}$, $P = \{p_0, p_1 \ldots p_r\}$ and $L = \{l_0, l_1, l_2 \ldots l_t\}$ are the sets of sections, points and lights, respectively.

Each section $s \in S$ has two ends denoted by *s.left* and *s.right* and each point p has three ends *p.up, p.down, p.main*. We shall use the notation $S^{ends} = \{s.left,\ s.right \mid s \in S\}$ and $P^{ends} = \{p.up,\ p.down,\ p.main \mid p \in P\}$. The injective function $\gamma : L \to S^{ends}$ assigns each light to a section. A light at a left (right) end of a section can only control a train moving on that section in the direction right to left (left to right). Finally, the connectivity of the sections and points is represented by undirected edges E such that $E \subseteq [V]^2$ where $V = P^{ends} \cup S^{ends}$ and $[V]^2 = \{Z \subseteq V \mid |Z| = 2\}$) satisfying

- if $e_1 \neq e_2$ then $e_1 \cap e_2 = \emptyset$ for all $e_1, e_2 \in E$,
- $\{s.left, s.right\} \notin E$ for each section $s \in S$, and
- for each $p.z \in P^{ends}$ there is an $s.x \in S^{ends}$ such that $\{p.z, s.x\} \in E$.

In other words, each end can connect to at most one other end, we allow point ends to connect only to section ends and section-loops as well as isolated point ends are not allowed.

2.2 Untimed Semantics of Railway Games

Consider a railway topology $R = (S, P, L, \gamma, E)$. We shall define its associated game graph for a given number of trains $T = \{t_0, t_1 \dots t_{k-1}\}$. The vertices of the game graph consist of configurations where each train is located at a section end and has a direction in which it is moving. For each point $p \in P$, a mode *up*/*down* is associated with it in the configuration, indicated by *p.mode*. Finally, for each light $l \in L$, there is a colour *red*/*green* associated with it in a given configuration. The underlying semantics is given as a game graph $\mathcal{G}_R = \langle C, \longrightarrow, \dashrightarrow, c_0, Bad, Goal\rangle$ defined in the rest of this section.

Configurations. The set of configurations is $C \subseteq (S^{ends} \times \{left, right\})^T \times \{up,\ down\}^P \times \{red,\ green\}^L$ so that a configuration $c \in C$ is a triple of three functions of the form (pos, mod, col) where

- $pos : T \to S^{ends} \times \{left, right\}$ stores the location and direction of each train $t \in T$. For each train $t \in T$, $pos(t)$ is an ordered pair of the form $(pos^1(t), pos^2(t))$ where $pos^1(t)$ indicates the section end in which t is currently located and $pos^2(t)$ indicates the train direction,
- $mod : P \to \{up, down\}$ records the mode of each point, and
- $col : L \to \{red, green\}$ remembers the current lights setting.

We assume a given initial configuration c_0 with the initial placement of all trains and a fixed color lights and modes of points, as well as a set of goal configurations *Goal* where all trains are in their target sections and the positions of points and light colors can be arbitrary. The set *Bad* contains all configurations where two trains are located on the same sections, formally $Bad = \{(pos, mod, col) \in C \mid$ there is $t, t' \in T$ where $t \neq t', pos^1(t) = s.x$ and $pos^1(t') = s.y$ for some $x,y \in \{left,right\}\}$.

Transitions. We shall first define a function $nextSec : \big(S^{ends} \times (P \to \{up, down\})\big) \to S^{ends}$ that, given the modes of points, determines what are the neighbouring sections in the network (we assume that $x,y \in \{left,right\}$ and $z \in \{main, up, down\}$):

- $nextSec(s.x, mod) = s'.y$ if $\{s.x, s'.y\} \in E$
- $nextSec(s.x, mod) = s'.y$ if there exists a $p \in P$ such that $\{s.x, p.z\} \in E$ and
  - $z =main$ and $mod(p) =up$ such that $\{p.up, s'.y\} \in E$, or
  - $z =main$ and $mod(p) =down$ such that $\{p.down, s'.y\} \in E$, or
  - $z =down$ or $z =up$ such that $\{p.main, s'.y\} \in E$
- $nextSec(s.x, mod) = s.x$ otherwise.

Let $c = (pos, mod, col)$ be a configuration from C. We shall also define a set of movable trains $movableTrains(pos, col) \subseteq T$ that, given the position of trains and colours of all lights, returns the set of all trains (if any) that are at the end of their sections and their corresponding lights (if any) are green in colour. Formally, $movableTrains(pos, col) = \{t \in T \mid pos(t) = (s.x,\ x)$ where $x \in \{left,right\}$ and $col(l) = green$ for all $l \in L$ such that $\gamma(l) = s.x\}$.

We are now ready to define the controllable and environmental transitions in the graph.

Controllable Transitions: These are transitions modelling the moves made by the controller in the game. Whenever there is a train at the end of a section, the controller can change the modes of points and the colours of lights, with the restriction that if a train is moving in a section where there is green light, it is not allowed to suddenly change it to red (as instanteniously stopping a moving train is not a realistic behaviour). The controller move is finished by placing all movable trains to the connected sections, unless there is a train that can crash to another train, in which case this train moves alone and the crash is detected by the fact that the target configuration belongs to the set *Bad*. Formally, we write $(pos, mod, col) \longrightarrow (pos', mod', col')$ if

- there exists a $t \in T$ such that $pos(t) = (s.x, x)$ where $x \in \{left,right\}$,
- for every $t \in T$ where $pos(t) = (s.y, x)$ for $x, y \in \{left,right\}$ if for every light $l \in L$ where $\gamma(l) = s.x$ holds $col(l) = green$ then $col'(l) = green$,
- and moreover
  - if there is $t \in movableTrains(pos, col')$ s.t. $nextSec(pos^1(t), mod') = pos(t')$ for some $t' \in T \setminus \{t\}$ then $pos'(t) = \Big(nextSec\big(pos^1(t), mod'\big), pos^2(t)\Big)$ and for every other $t' \in T \setminus \{t\}$ we have $pos'(t') = pos(t')$,
  - otherwise $pos'(t) = \Big(nextSec\big(pos^1(t), mod'\big), pos^2(t)\Big)$ for every $t \in movableTrains(pos, col')$ and $pos'(t') = pos(t')$ for every other t'.

Environmental Transitions: Finally, we can define the transitions controlled by the environment modelling the uncertainty whether the trains move along the sections and what set of trains move concurrently. We define $(pos, mod, col) \dashrightarrow (pos', mod, col)$ if

- $movableTrains(pos, col) = \emptyset$,
- for every $t \in T$ either $pos'(t) = pos(t)$, or $pos'(t) = (s.y,y)$ provided that $pos(t) = (s.x,y)$ for $x, y \in \{left,right\}$), and
- $pos \neq pos'$.

The first condition guarantees that if there are some movable trains at the end of the sections, then they move to their neighbouring sections (by means of controllable transitions). The second condition gives the environment the freedom to decide any subset of trains that (concurrently) arrive at the ends of their respective sections and the last conditions guarantees that at least one train moves in order to guarantee progress (we want to avoid environmental self-loops as the game will not have any winning strategy in this case). Notice that the environment cannot influence the modes of points nor the setting of lights.

Example 1. The railway network shown in Figure 1 is in the initial configuration $c_0 = (\mathit{pos}, \mathit{mod}, \mathit{col})$ where $\mathit{pos}(t_0) = (s_1.\mathit{left}, \mathit{right})$, $\mathit{pos}(t_1) = (s_3.\mathit{right}, \mathit{left})$, and $\mathit{mod}(p_0) = \mathit{mod}(p_1) = \mathit{up}$, $\mathit{mod}(p_2) = \mathit{mod}(p_3) = \mathit{down}$, and say that $\mathit{col}(l_0) = \mathit{col}(l_1) = \mathit{col}(l_2) = \mathit{red}$. There are no controllable transitions in c_0 as no trains are at the ends of the sections, i.e., $\mathit{movableTrains}(\mathit{pos}, \mathit{col}) = \emptyset$. However, the environment can move either the train t_0 from $s_1.\mathit{left}$ to $s_1.\mathit{right}$ or the train t_1 from $s_3.\mathit{right}$ to $s_3.\mathit{left}$, or both of them at the same time. Suppose it is the second case and the train t_1 arrives at $s_3.\mathit{left}$. Now the controller can swap the mode of p_0 and set the light l_2 to green, which implies that the train t_1 moves to $s_0.\mathit{right}$. Now it is the environmental turn and say that the train t_0 arrives to $s_1.\mathit{right}$. The controller can safely set the light l_1 to green and without any further control, also the train t_0 eventually arrives to its target section s_6.

2.3 Stochastic Semantics for Railway Games

In the railway game provided in the previous section, the movement of trains along sections has been purely discrete with no information about the timing of these movements. In this section we refine this view by assuming that the time it takes a train t to pass a section s is given by a distribution $\mu_{t,s}$. Here we shall assume that the passage-time distributions are given by exponential distributions. Choosing exponential distributions simplifies the technical presentation due to the memoryless property of exponential distributions, however, in Uppaal Stratego there is a support for several other distributions as well as for the possibility to make the distribution parameters depend on weather conditions and other external factors (see [7]). However, assuming only knowledge about the expected passage-time, exponential distribution is anyway the most appropriate choice in terms of entropy.

A stochastic railway game for a set of trains T is a tuple (S, P, L, γ, E, R), where (S, P, L, γ, E) is a railway game and $R : T \times S \to \mathbb{R}_{\geq 0}$ provides for each train $t \in T$ and each section S the rate $R(t, s)$ of an exponential distribution being the passage-time distribution $\mu_{t,s}$. We recall that for an exponential distribution with rate r, the density of passage-time d is $r \exp^{-r \cdot d}$, and the probability that the passage-time will be less than d is $1 - \exp^{-r \cdot d}$. Also, the expected passage-time is $\frac{1}{r}$. Finally, given two trains t_1 and t_2 with passage-time rates r_1 and r_2, the probability that t_1 completes its passage first is $\frac{r_1}{r_1 + r_2}$.

In the full railway game, various trains (all the ones that are not stopped by a red light at the end of a section) are independently moving along different sections simultaneously with the passage-times given by exponential distributions with rates prescribed by R. One of these trains will reach its end first[1]. This calls for a stochastic refinement of the uncontrolled train transitions of the railway game. Consider the untimed train transition $(pos, mod, col) \dashrightarrow (pos', mod, col)$, where train t – non-deterministically choosen between the moving trains – is the unique train reaching the end of its section s, i.e. $pos(t) \neq pos'(t)$. In the stochastic refinement we will assign a density δ for this transition happening at time d. Now let $M \subseteq T$ describe the set of trains moving excluding the winning train t. Also for $t' \in M$, let $s(t')$ denote the section along which t' is moving. Then a timed train transition is of the form:

$$(pos, mod, col) \dashrightarrow^{d}_{\delta} (pos', mod, col)$$

where d is a passage-time and δ is given by:

$$\delta = R(t,s) \cdot \exp^{-R(t,s)\cdot d} \cdot \prod_{t' \in M} \exp^{-R(t',s(t'))\cdot d}$$

In the above the first two terms of the product – $R(t,s) \cdot \exp^{-R(t,s)\cdot d}$ – is the density that train t passes section s in d time-units. However, the density δ of the train transition must also reflect that t is the first train to reach the end of its section. This is expressed by the last product term. Note that $\exp^{-R(t',s(t'))\cdot d}$ is the probability that train t' has not completed the passage of its section $s(t')$ in d time-units. Due to the assumed independence of the passage-times of trains, the product among all these equals the probability that no other moving train but t has reached the end of its section before d.

The above notion of density of a timed train transition extends to densities on finite timed runs by simple multiplication of the densities of the timed train transitions appearing in the run. Now constrained by a strategy σ, the railway game becomes fully stochastic as the non-deterministic choices of the controller are resolved by the strategy. Hence – by integration and addition – the densities on runs determine a probability measure $\mathbb{P}_\sigma$ on sets of outcomes under σ. In particular, for a given strategy σ we may determine the probability of the set of runs leading to a crash of two trains. If no crash under the strategy can occur, we may determine the expected time until all trains have reached their goal.

Example 2. Reconsider the railway network from Figure 1. Assume that all lights are green (sounds dangerous, and it is!). A possible control strategy σ_1 could try to avoid disaster by turning point p_2 up once any of the two trains reached the end of the initial section. However, there is still a possibility of crash as train t_1 may complete both sections s_3 and s_7 before train t_0 completes section s_1. The

[1] The event that two or more trains reach the end of their sections simultaneously has measure zero and may be ignored.

following shows that the probability of this is $\frac{2}{9}$.

$$\begin{aligned}
\mathbb{P}_{\sigma_1}(Crash) &= \mathbb{P}_{\sigma_1}(t_1 \text{ completes } s_3 \text{ and } s_7 \text{ before } t_0 \text{ completes } s_1)\\
&= \mathbb{P}_{\sigma_1}(t_1 \text{ completes } s_3 \text{ before } t_0 \text{ completes } s_1) \cdot\\
&\quad \mathbb{P}_{\sigma_1}(t_1 \text{ completes } s_7 \text{ before } t_0 \text{ completes } s_1)\\
&= \frac{\frac{1}{4}}{\frac{1}{4}+\frac{1}{2}} \cdot \frac{\frac{1}{1}}{\frac{1}{1}+\frac{1}{2}} = \frac{1}{3} \cdot \frac{2}{3} = \frac{2}{9}
\end{aligned}$$

Example 3. Again consider the railway network from Figure 1. In this scenario we assume that all lights are initially red (sounds better from a safety point of view). Here we consider a safety strategy σ_2, where whenever a train (t_1 respectively t_0) reaches its end the corresponding light (l_2 respective l_1) is turned green and at the same time the corresponding point (p_0 respective p_2) is moved (down respectively up). This strategy guarantees safety, so the probability of the two trains crashing is 0. The expected time until both trains are at their goal location under σ_2 is 13 as seen by:

$$\begin{aligned}
\mathbb{E}_{\sigma_2}[Goal] &= \max\big\{\mathbb{E}_{\sigma_2}[t_1 \textit{ in goal}], \mathbb{E}_{\sigma_2}[t_0 \textit{ in goal}]\big\}\\
&= \max\{4+2, 2+2+7+2\} = 13
\end{aligned}$$

Example 4. As a final example, consider yet again the railway network from Figure 1. Let us first consider that t_0 is the first train to reach the end of its section. The optimal strategy σ_o for the controller is now to move p_2 up, and drive straight for the goal, with an expected time of 13. If, however, t_1 is the train to reach its end first, t_0 can move through the sections s_7, s_3, s_8, with an expected time of 8. The expected time for both trains to reach their goal under this strategy is 12.3, as calculated below.

$$\begin{aligned}
\mathbb{E}_{\sigma_o}[Goal] &= \max\{\mathbb{E}_{\sigma_o}[t_1 \textit{ in goal}], \mathbb{E}_{\sigma_o}[t_0 \textit{ in goal}]\}\\
&= \max\big\{4+2, \mathbb{P}_{\sigma_o}(t_0 \text{ completes } s_1 \text{ before } t_1 \text{ completes } s_3)\\
&\qquad \cdot(2+2+7+2) +\\
&\qquad \mathbb{P}_{\sigma_o}(t_0 \text{ completes } s_1 \text{ after } t_1 \text{ completes } s_3)\\
&\qquad \cdot(2+1+4+1+2)\big\}\\
&= \max\big\{4+2, \frac{7}{9} \cdot 13 + \frac{2}{9} \cdot 10\big\} = 12.3
\end{aligned}$$

3 Railway Games in Uppaal Stratego

After having introduced the theoretical foundations of our untimed railway game and its stochastic extension, we shall discuss the encoding of our approach into timed automata in the Uppaal-style and use the tool Uppaal Stratego [6] to synthetise (near) time-optimal and safe control strategies.

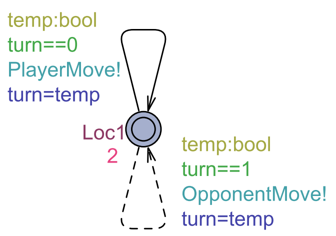


Fig. 2. An example of a timed game automaton of the tool UPPAAL STRATEGO

3.1 Translation to Timed Game Automata

UPPAAL STRATEGO uses timed game automata as models, which are an extension of timed automata [2]. Timed automata extend finite state machines with a number of real-valued clocks that enable them to measure the progress of time. The automata used by UPPAAL are extended further, by allowing for additional model features, e.g. C-like syntax, explained below. Timed game automata additionally divide transitions into controllable and uncontrollable transitions. Figure 2 provides an example of a timed game automaton in UPPAAL STRATEGO, containing all feature used in the presented models. The automaton consists of one location (*Loc1*), which contains an exponential rate (*2*) determining how long we have to stay in that location before performing a transition. The transition on top is controllable, denoted by the solid line. The controller can only execute the transition, if the turn variable is currently 0. The transition sends a signal (*PlayerMove!*) to other automata (run in parallel and not showed in the figure) when executed. Finally, the controller can choose a value for the variable *temp*, which will be assigned to the global variable *turn*. If it assigns the value 1, its the opponent's turn, which means it can execute the transition below (dotted line). Before such a transition is executed, we again delay according to the exponential rate of the location, and the opponent can choose to keep broadcasting on the channel *OpponentMove!* until it decides to change the value of *turn*.

When encoding our railway games into UPPAAL STRATEGO, we applied two main optimizations in order to reduce the state space of the games. One opti-

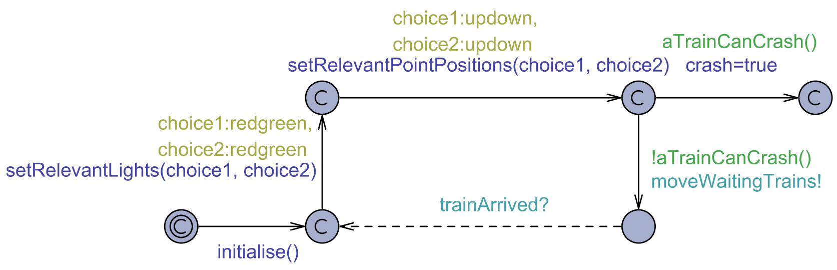


Fig. 3. Controller of a railway game modelled as a timed automaton

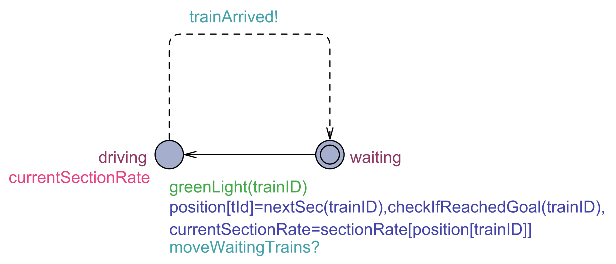

trainArrived!
driving
currentSectionRate
waiting
greenLight(trainID)
position[tId]=nextSec(trainID),checkIfReachedGoal(trainID),
currentSectionRate=sectionRate[position[trainID]]
moveWaitingTrains?

Fig. 4. A train in a railway game modelled as a timed automaton

mization was inspired by [12], and it separates lights and points into relevant and irrelevant sets. Only lights that are on sections currently occupied by a train are considered relevant for the controller. The same applies to points, that is, only points connected to occupied sections are relevant. When the controller moves, it can only change the configuration of currently relevant lights/points. All other lights/points are set to their default values (red/down). The second optimization is a static analysis of the railway network. For each train we perform an exploration of the connectivity in railway network and collect all the sections it may visit in order to be still able to reach its target section. If a mode of a given point leads to a section from which it is not feasible to move the train to its target location, we remove this mode choice from the controller for this specific train. This further reduces the state-space that UPPAAL STRATEGO needs to explore during the controller synthesis. Both optimizations do not change the existence of winning strategies nor remove any time-optimal strategies. Another optimization was to enforce green light for at least one train, every time the controller sets the lights. This will ensure progress by forcing at least one train to move at any given point.

Figures 3 and 4 show a (simplified) example of the used timed automata models[2]. The models are split into one automaton for the controller, and one automaton for each train. Only one template of the trains is shown. In addition, the UPPAAL files contain C-line code, including variables for storing the layout of the stations, and the current configuration.

In Figure 3 we show the controller: the controller is responsible for initializing the railway station (setting all the variables for the layout), before the game starts. After that, it can chose the mode of the relevant lights, setting them to either green or red. Then it can set the mode of the relevant points. In this example, we consider only two trains, hence only two lights/points need to be set here. After setting the lights and points, if there is a possibility for a crash, the controller sets the crash variable to true. Note that if the controller has a winning strategy, this will never occur. If no train can crash, all trains that are waiting for green light are notified that the configuration changed. When these

[2] Our experiment files can be found online at `http://people.cs.aau.dk/~florber/TrainGames/ExperimentFiles.rar`

steps are done, the controller waits for a train to arrive at the end of a section, at which point it will be able to change the settings again.

Figure 4 illustrates a train: initially, trains are starting at the end of a section, waiting to pass to the next section. If the train receives the signal that the configuration was changed (*moveWaitingTrains?*) and is has green light now, it can move to the next section and start driving there. When a train moves to the next section, several variables are updated. First, the train updates its current position according to the *nextSec* function, then it checks whether it reached its final destination, and finally it updates its rate to the rate of the section it is currently driving on. If it reaches the end of the section, it will signal this to the controller, which is done via sending the *trainArrived!* signal. The duration until a train reaches the end of a section is given by the rate of the current section, which in the figure is illustrated next to the driving location.

In Figure 5 is an excerpt of the *nextSec* function implemented in case of the railway network shown in Figure 1 which either reports a crash or returns the next section end of the train, depending on the settings of points and lights. For example, if the current position of the train is s_1*.right* (stored in the variable *pos*), the train is going towards the right, the light l_1 is green and point p_2 is in *down* mode, the train ends up in s_7*.left*. Now if s_7 is already occupied then it results in a crash otherwise the position of the train is updated accordingly.

```
section_End nextSec(int tId){
⋮

pos=currentPosition[tId];
if(pos == s1.right and dir == right)
 if(colour[1] == green)
    if(mode[2]= down)
       if(sectionOccupied[7] == false) pos=s7.left;
       else crash=true;
   else
      if(sectionOccupied[2] == false) pos=s2.left;
      else crash=true;
 else
   pos=s1.right;
⋮

return pos;
}
```

Fig. 5. Fragment of Uppaal C-code for changing a train position

3.2 Uppaal Stratego

Uppaal Stratego [6] combines the two branches Uppaal TIGA [4] and Uppaal SMC [7]. Uppaal TIGA provides an on-the-fly algorithm for synthesis

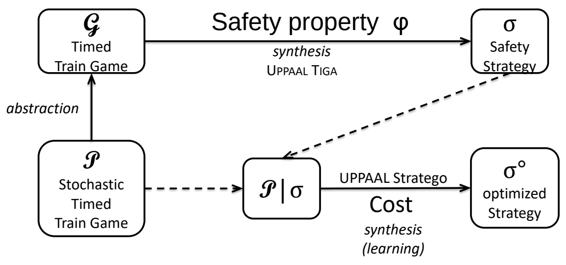


Fig. 6. Overview of the functionality of Uppaal Stratego

of reachability and safety objectives for timed games. Uppaal SMC provides statistical model-checking for stochastic timed games. The workflow of Uppaal Stratego which combines the two tools can be seen in Figure 6. To run Uppaal Stratego, one has to define queries for the model checker. We used two types of queries to generate the strategies.

The query below synthesises the safe strategy. It asks the model checker for a control strategy, such that the Boolean variable crash is always false.

```
strategy safe = control:A[] not crash
```

The optimization of our strategy with machine learning requires the specification of a cost function. In our case, the cost is given by the time passing, while there is a reward for bringing a train to its destination. We simulate for at most 151 time units (or until all trains exited), thus we set the reward to 150, such that exiting the train is always of higher priority than ending fast. This optimization is performed while adhering to the safe strategy. The query is given below.

```
strategy opt =  minE  (time - 150*(exited[0])) [<=151]:
<> (time>=150  ||  exited[0]) under safe
```

The second step does not provide necessarily the time-optimal strategy (as the problem is in general undecidable), however, by the use of reinforcement learning it approaches the optimal solution and works convincingly in practice.

4 Experiments

We shall now present the experiments and results we achieved using Uppaal Stratego. We analyzed two Danish railway stations, Aalborg and Lyngby. The

Station	# Sections	# Lights	# Points	# Trains
Lyngby	11	14	6	2 to 5
Aalborg	26	41	14	2 to 5

Table 1. Specifics about the Lybgy and Aalborg station

Station	2 Trains	3 Trains	4 Trains	5 Trains
Lyngby	0.04	0.16	3.37	55.93
Aalborg	0.24	16.57	858.91	timeout

Table 2. Time (in seconds) for computing the most permissive safe strategy

railway layout for Lyngby was based on the layout presented by Kasting et al. [12] that we extended by the rates associated to each section. The layout used for Aalborg is based on a track plan found online [1] where we considered the main tracks, disregarding the cargo tracks. We assumed that each entry to a point is guarded by a light. The rates for both stations were estimated by using Google Maps to figure out the length of the sections, and assuming that trains drive with $80kmh$ outside of the station, and $40kmh$ in the proximity of the platforms. To achieve that all rates are higher than 1, the rates of Lyngby were multiplied by 66, and all rates in Aalborg by 20. Thus, one time unit in an experiment with Lyngby/Aalborg below corresponds to 66/20 seconds, respectively.

4.1 Setup

We considered the problem with 2, 3, 4 and even 5 trains concurrently moving at the station in order to explore the scalability of our approach. The initial and final locations for the trains in the Lyngby station were chosen similarly to [12]. The trains in Aalborg are placed in a way to make their travel as complex as possible, i.e., they always have to travel from one side to the other, taking several points and cross each others paths. The models with less than 5 trains were produced by removing trains from the complete model with a maximum number of trains. The specifics about the stations can be found in Table 1, showing that Aalborg is about twice the size of Lyngby. The experiments were executed on AMD Opteron 6376 processor running at 2,3Ghz with 10GB memory limit.

4.2 Results

In Table 2 we report on the time to compute the safety strategy. The runtime for computing the strategy depends on the size of the station and, as expected, it grows exponentially with the number of trains (Aalborg with 5 trains timed out). This means that for this high number of trains, the proposed approach is at the moment infeasible without further state-space reductions. However, even at the larger stations like Aalborg, in reality there should rarely be more than three trains approaching the station at the same time.

In Table 3 we investigate how Uppaal Stratego performs in optimizing the constructed safe strategy. We use a currently unpublished learning method of Uppaal Stratego relying on Q-learning [15]. We use 50 iterations and each of 1000 runs for learning the strategies for Lyngby, and 5000 runs for Aalborg, as the increased state-space requires a higher learning effort. We report the time needed to produce the safe strategy plus to its optimization in Uppaal Stratego, the expected time until all trains reach their goals under the synthesised

Station	Aal2	Aal3	Aal4	Aal5	Lyn2	Lyn3	Lyn4	Lyn5
UPPAAL runtime	650.53	1501.46	3990.49	—	66.79	143.86	244.38	382.33
Expected time	1.8	2.62	2.65	—	1.06	2.6	2.27	2.72

Table 3. Strategy optimization, including the runtime of safe strategy synthesis (in seconds), and the expected time until all trains reach their destinations

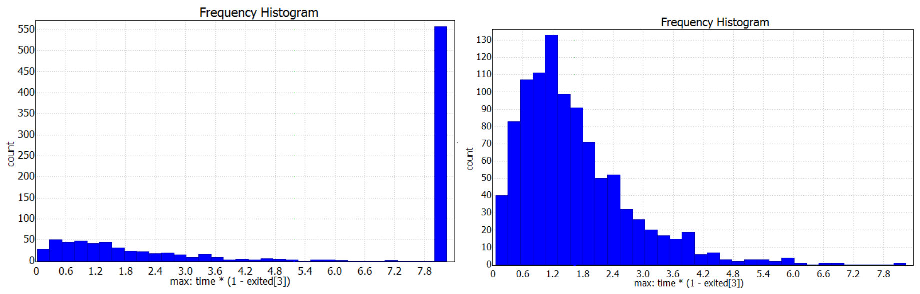


Fig. 7. 1000 random runs on Lyngby, the right plot is under the optimized strategy

strategies where e.g. Aal3 means the Aalborg station with 3 trains. The expected time was computed by simulating 2000 random runs under the strategy. The presented values are an average computed from repeating the experiments 20 times. The stochastic nature of the trains can of course influence the observed timed behaviour, and lead to small time fluctuations in the expected time.

Figure 7 shows the frequency histogram of the arrival times of the train number 4 during 1000 simulations on Lyngby with 5 trains, where the unoptimized safe strategy is on the left side and the guided optimized strategy is on the right side. The x-axis represents the train arrival time in minutes and the y-axis the number of times the train arrived at the given time. Clearly, the unguided strategy has a majority of simulation where the train did not arrive within 8 minutes, whereas the optimized strategy makes the train number 4 to arrive on average in 1.6 minutes.

5 Conclusion

We presented a game-theoretic approach for controlling of a railway network as a two player game between the controller (setting up the lights and modes of points) and the environment (moving the trains). Our approach guarantees safety (absence of trains simultaneously entering the same section) by computing the most permissive safe strategy in the untimed game. This strategy is further optimized in the model enhanced with stochastic semantics, approximating the time trains use to travel across a section, in order to optimize the speed of trains arriving to their target sections.

The main novelty of the proposed approach is the support for concurrently moving trains and the synthesis of (near) time-optimal controllers that are safe.

Both these steps can be automatically realized in our tool Uppaal Stratego. This is an important step towards reflecting the behaviour of trains in reality. Our approach was demonstrated and evaluated on two Danish railway stations, using different number of trains. The experiments clearly show the feasibility of our approach, however, for one of the stations, the highest number of trains led to a timeout, highlighting also the limitations of our current implementation.

In the future work, we shall work on improving the performance of our tool and on applying reduction techniques in order to decrease the size of the state-space, similarly as it was done in [10] for the untimed and strictly alternating game. We will also look at the scheduling problem where each train has a predefined route like in [11]; this is a realistic assumption and it will likely reduce the complexity of the control synthesis. Furthermore, we can without any effort use cost functions of different types e.g. to prioritize or penalize certain trains depending on their importance. For further evaluation of our strategies, we plan on making it easier to extract them from our tool, so that they can be applied to simulations in other software or control directly model trains in a demonstrator that we plan to build. Finally, in our railway model we made a few simplifying assumptions that we plan to relax in our future work. For example, we shall add a travel time through a point (at the moment we assume that it is instantaneous) and make sure that points are not operated while trains are passing over them in order to prevent derailment.

Acknowledgments. We would like to thank to Peter G. Jensen for his support with the experiments and advice on Uppaal Stratego. We also thank the anonymous reviewers for their detailed comments and in particular for pointing out a problem in our original formal model that could have made the constructed controller potentially unsafe. The research leading to these results has received funding from the project DiCyPS funded by the Innovation Fund Denmark, the Sino Danish Research Center IDEA4CPS and the ERC Advanced Grant LASSO. The fourth author is partially affiliated with FI MU, Brno, Czech Republic.

References

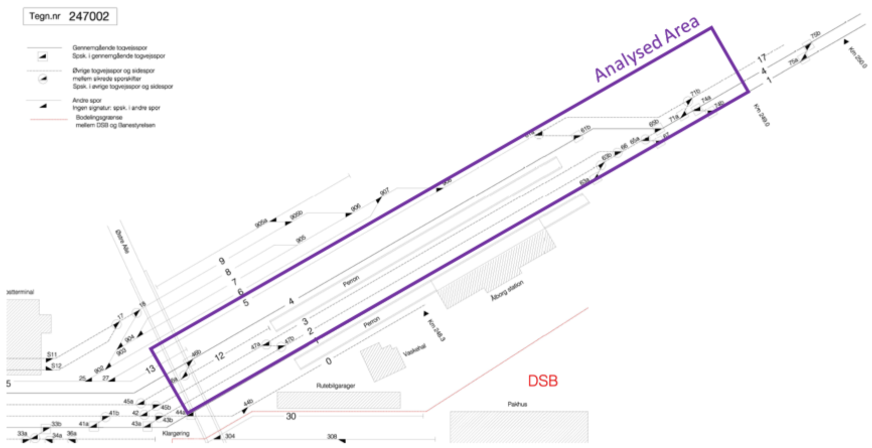
1. Danish railway station plans. `https://www.sporskiftet.dk/wiki/danske-spor-og-stationer-sporplaner-og-link/`. Accessed: 2019-01-14.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994.
3. B. Aristyo, K. Pradityo, T. A. Tamba, Y. Y. Nazaruddin, and A. Widyotriatmo. Model checking-based safety verification of a petri net representation of train interlocking systems. In *2018 57th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pages 392–397, Sep. 2018.
4. Gerd Behrmann, Alexandre David, and Kim G. Larsen. *A tutorial on uppaal*, pages 200–236. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
5. A. Cimatti, F. Giunchiglia, G. Mongardi, D. Romano, F. Torielli, and P. Traverso. Formal verification of a railway interlocking system using model checking. *Formal Aspects of Computing*, 10(4):361–380, Apr 1998.
6. Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikučionis, and Jakob Haahr Taankvist. Uppaal stratego. In Christel Baier and Cesare Tinelli,

editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 206–211, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

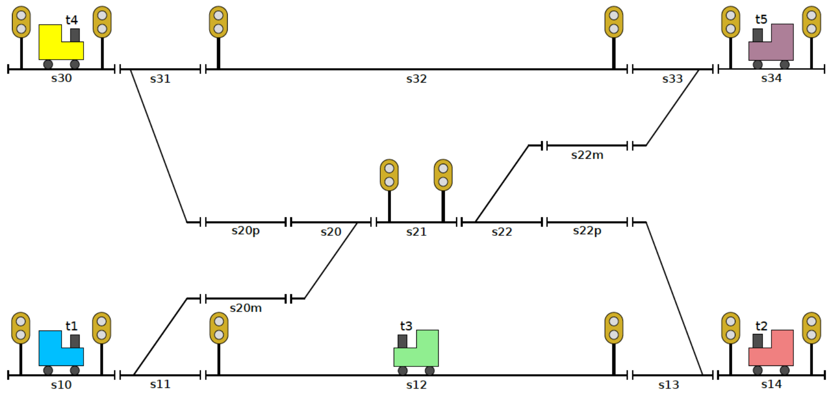
7. Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, Aug 2015.
8. Andreas Berre Eriksen, Chao Huang, Jan Kildebogaard, Harry Lahrmann, Kim G Larsen, Marco Muniz, and Jakob Haahr Taankvist. Uppaal stratego for intelligent traffic lights. In *12th ITS European Congress*, 2017.
9. A. Giua and C. Seatzu. Modeling and supervisory control of railway networks using petri nets. *IEEE Transactions on Automation Science and Engineering*, 5(3):431–445, July 2008.
10. Michael R Hansen. On-the-fly solving of railway games (work in progress). *Marina Waldén (Editor)*, page 34, 2017.
11. Anne Elisabeth Haxthausen. Automated generation of formal safety conditions from railway interlocking tables. *STTT*, 16(6):713–726, 2014.
12. Patrick Kasting, Michael R. Hansen, and Steen Vester. Synthesis of railway-signaling plans using reachability games. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages*, IFL 2016, pages 9:1–9:13, New York, NY, USA, 2016. ACM.
13. Kim Guldstrand Larsen, Marius Mikučionis, and Jakob Haahr Taankvist. *Safe and Optimal Adaptive Cruise Control*, pages 260–277. Springer International Publishing, Cham, 2015.
14. Jakob Lyng Petersen. Automatic verification of railway interlocking systems: A case study. In *Proceedings of the Second Workshop on Formal Methods in Software Practice*, FMSP '98, pages 1–6, New York, NY, USA, 1998. ACM.
15. Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.
16. Kirsten Winter. Model checking railway interlocking systems. In *Proceedings of the Twenty-fifth Australasian Conference on Computer Science - Volume 4*, ACSC '02, pages 303–310, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

Appendix

Layout of Aalborg, taken from [1].



Layout of Lyngby, taken from [12].



t4
t5
s30
s31
s32
s33
s34
s22m
s20p
s20
s21
s22
s22p
s20m
t1
t3
t2
s10
s11
s12
s13
s14