Evaluating regulation policies for subways with model checking

Benjamin Bordais Thomas Mari Julie Parreaux

Abstract

Subway rail systems are particularly sensitive to unexpected delays or random events. To any subway rail system, one can associate a regularity (for instance, a train arrives at a station every four minutes) that the system needs to ensure. To stick to this expectation, rail systems are equipped with regulation policies that give instructions in real time when changes occur in the traffic. In this report, we consider how tools such as model checkers can help in the design and evaluation of regulation policies. Usually, these tools work with formal models. Therefore, our purpose is to properly design a model of a rail system and to analyze regulation policies on this model.

Keywords : Subway rail system; Model checking; Abstract model.

I. Introduction

Subway rail systems are expected to meet some requirements like safety or regularity of service. Ideally, trains arrive and leave stations exactly when they are supposed to so that the system ensures a regular service (e.g., a train arrives at a station every four minutes). However, it usually is not the case since random disturbances occur frequently (for example, delays due to passengers misconducting). To stick to the expected service, rail systems are equipped with regulation policies that give instructions when changes occur in the traffic. Instructions can be, for instance, to increase or decrease the dwell time in station.

The study of rail systems can be divided into two orthogonal concerns. The first one consists in the study of rail systems to prove that they ensure some safety measures. Safety of trains is usually ensured by critical sections (track portions that can be entered by at most one train), that are physically implemented by signals[1] and interlocking systems[2]. In both cases, automated verification is used since formal guarantees are needed for safety measures. More specifically, the second paper uses bounded model checking to achieve the verification objectives. It has to be noted that these verifications are done independently of any regulation policy.

Once it is proven that the rail system ensures the safety measures, one can study the performance of regulation policies regarding delays in that system. This can be done by using automatic tools, which can be divided into the microscopic and macroscopic ones (as suggested in[3]). Basically, macroscopic tools use abstract models without taking into account details like the adherence to tracks or the passenger flow. NEMO[4] is an example of such a tool. On the other hand, the microscopic approach takes into account every detail of the rail system including, for instance, weather conditions. Then, they simulate the evolution of a network during a time period, considering the evolution of trains between time steps of fixed length (typically, one second). OpenTrack is a tool implementing this approach[3]. However, simulating by micro-steps is very time and space consuming. Moreover, such an approach does not allow to use model checking since the model of the system that is considered is too complex.

The microscopic and macroscopic approaches can also be used more specifically for subway rail systems. For instance, the SimMETRO[5] tool uses the microscopic approach for subway system. The macroscopic approach is used in the paper [6] in which Petri Nets are used to model rail system. However, since only simulation is used in these methods, the formal guarantee about the accuracy of the results obtained are statistical.

Our goal is to evaluate the performance of regulation policies in subway rail systems and we also want formal guarantees on our results. That is why we will use model checking to evaluate our regulation policies. Basically, given a mathematical model and a logical formula describing the requirements, model checking checks if the model satisfies the formula. Since subway systems are complex, model checking capabilities may be overwhelmed quickly. That is why we will use a macroscopic approach, to work on a model as small as possible.

First, we need a model that depicts rail systems on which a model checker can work. We need to have in our model both probabilities (to depict the randomness of delays) and nondeterminism (to depict the choice drivers or regulation policy can make to increase or decrease their dwell time in station). A popular model that contains both probabilities and nondeterminism is Markov Decision Process [7] (MDP for short). So, rail systems will be modeled as MDPs. MDP is a discrete model, which means that, we will need to discretize time and space. We assume that safety properties are ensured by the system. That is why we will make the hypothesis that no train can collide in the system we model. That means that the MDP we will design to depict the real rail system needs to forbid trains to collide. To our knowledge, this approach has not been studied before.

Once our model is designed, we will be able to design a regulation policy. Then, we will need a logic to express the formulas of interest for us. Such formulas would express, for example, that if a train is delayed, with a high probability, at some point in the future it will be on time. The most suitable logic to express this kind of properties is the Probabilistic Computational Tree Logic [8] [9] (PCTL in short).

The automatic tools we use for verification are model checkers. The model checker we will use is PRISM [10], since it can work on MDPs and the PCTL logic. Moreover, PRISM can be used for exact model checking which yields an exact result provided that it can effectively build the model (that is, enumerate every state and transitions) but also for statistical model checking. Statistical model checking can be used even when the model is too big to be built in PRISM, but it yields a result with (a formally quantified) uncertainty.

Finally, once we have designed a model, if it is too big for PRISM to handle and if we want an exact result on this model (for instance, to prove that this model respects our hypothesis which states that the system is safe), then we will need to reduce its size. That is why we often have to abstract our model [11] [12] of the rail system.

Our objective is to evaluate the performance of regulation policies on a real rail system. First, we need to build a model of the rail system. Then, once the soundness of the model is established (that

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0.7 & 0 & 0.3 \\ 0.5 & 0 & 0.5 \end{pmatrix}$$

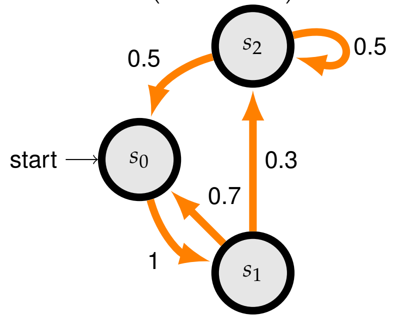

Figure 1. The DTMC $(\{s_0, s_1, s_2\}, s_0, P, \{a\}, L)$ with $L(s_0) = \{a\}$ and $L(s_1) = L(s_2) = \emptyset$

is, it respects our hypothesis), we can study regulation policies with PRISM.

This report is organized as follows: Section 2 provides the technical background necessary for the understanding of this article. Section 3 introduces our first approach of the problem and the initial MDP we considered, with a regulation policy. Then, Section 4 explains our second approach and the results we obtained. Section 5 concludes.

II. Technical background

A. MDPs : Markov Decision Process

We introduce the notion of Discrete-Time Markov Chain which can be seen as a directed graph, in which vertices represent the state of the system, and the transitions represent the probability to change states.

Definition 1. *A Discrete-Time Markov Chain (DTMC) is a Tuple $\mathcal{D} = (S, \bar{s}, P, AP, L)$ where*

- *S is a finite non-empty set of states,*
- *$\bar{s}$ is an initial state*
- *$P : S \times S \to [0, 1]$ is a transition probability matrix such that $\forall s \in S, \sum_{s' \in S} P(s, s') = 1$*
- *AP is a set of atomic propositions*
- *$L : S \to 2^{AP}$ is a labeling function such that $\forall s \in S, L(s)$ is the subset of AP assigned to s.*

Figure 1 represents a DTMC, the probabilities are given by the matrix P. $P(i, j)$ is the probability to move to state s_j when the system is in state s_i. These probabilities can be read directly on the vertices.

We introduce the notion of Markov Decision Process (MDP), which allows to represent the nondeterministic evolution of a system. In a given state, an agent can choose an action which yields a distribution of probabilities for the next transition.

Definition 2. *A Markov Decision Process (MDP) is a tuple $\mathcal{M} = (S, \bar{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, AP, L)$ where*

- *S, $\bar{s}$, AP and L are as for DTMCs.*
- *$\alpha_{\mathcal{M}}$ a finite set of actions*
- *$\delta_{\mathcal{M}} : S \times \alpha_{\mathcal{M}} \times S \to [0, 1]$ where $\delta_{\mathcal{M}}(s, \alpha, s')$ represents the probability of going into the state s' from the state s when action α is chosen. In addition, we have that for $s \in S$, and $\alpha \in \alpha_{\mathcal{M}}$, $\sum_{s' \in S} \delta_{\mathcal{M}}(s, \alpha, s') = 1$.*

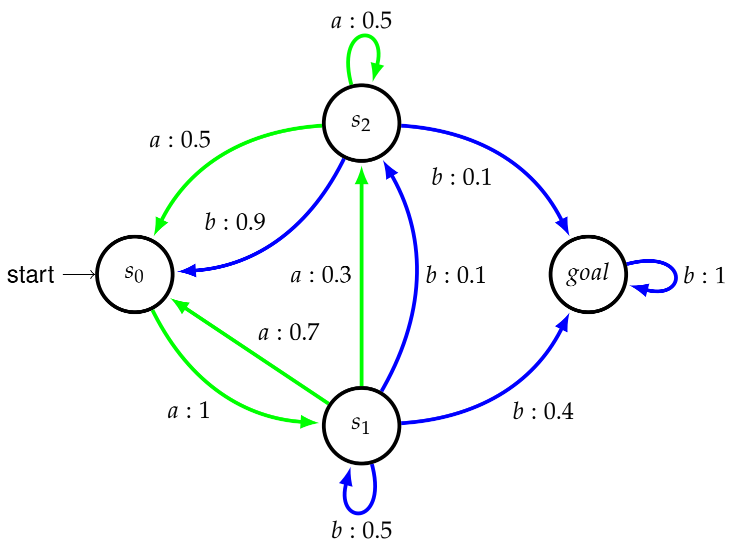

Figure 2. The MDP $(\{s_0, s_1, s_2, goal\}, s_0, \{a, b\}, \delta, \{a, goal\}, L)$ with $L(s_0) = \{a\}$, $L(s_1) = L(s_2) = \emptyset$ and $L(goal) = \{goal\}$ with $\delta(s, a, s') = P(s, s')$ (from the previous DTMC) and $\delta(s_1, b, s_1) = 0.5$, $\delta(s_1, b, s_2) = 0.1$, $\delta(s_1, b, goal) = 0.4$, $\delta(s_2, b, s_0) = 0.9$, $\delta(s_2, b, goal) = 0.1$

The actions are chosen nondeterministically. A policy allows to resolve the nondeterminism. At each state, a policy gives the action to take according to the states visited and the actions taken so far.

Definition 3. *We define $Path^{\mathcal{M}}$ as the set of all (infinite) sequences $s_1\alpha_1 s_2\alpha_2...$ such that $\forall i \geq 0$ $s_i \in S, \alpha_i \in \alpha_{\mathcal{M}}$ and $P(s_i, \alpha, s_{i+1}) > 0$. $Path^{\mathcal{M}}_{fin}$ is the set of finite prefixes of $Path^{\mathcal{M}}$ which terminate on a state. A policy is a function in $Path^{\mathcal{M}}_{fin} \to \alpha_{\mathcal{M}}$*

From an MDP $\mathcal{M}$ and a policy σ we can generate a DTMC $\mathcal{M}^{\sigma}$. However, it is not straightforward and we only present here the case where the policy is memoryless. That is, a policy σ such that $\forall m, m' \in Path^{\mathcal{M}}_{fin}$ with $last(m) = last(m')$ we have $\sigma(m) = \sigma(m')$ (the choice of the action to take only depends on the current state). We denote by $\sigma(s)$ the decision taken by σ when $s = last(m)$. Formally, with such a policy σ and an MDP $\mathcal{M} = (S, \bar{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, AP, L)$ we obtain the DTMC $\mathcal{M}^{\sigma} = (S, \bar{s}, P_{\sigma}, AP, L)$ with $P_{\sigma}(s, s') = \delta(s, \sigma(s), s')$. For example, the DTMC in Figure 1 can be generated from the MDP in Figure 2 and the (memoryless) policy σ such that $\forall m \in Path^{\mathcal{M}}_{fin}$, $\sigma(m) = a$.

B. Properties

Once our model is designed, we want to check whether the system satisfies some properties. One can distinguish situations that should always occur, and conversely situations that shall be avoided. Therefore properties are split into two categories. On the one hand, there are properties that express that a good event will happen. For instance, "if a train has some delay, it will recover from this delay eventually". On the other hand, there are properties that express that a bad event will not happen. The property "two trains will not collide" is a an example of such a property.

It has to be noted that, since our model is probabilistic, some properties we want to check on our model also need to be probabilistic. So, it is interesting to have answers that are not Boolean. That is why we want to be able to weigh our properties by probabilities.

Now, we need a logic to express these properties. In this logic, we

$$\Phi ::= \ \textbf{true} \mid c \mid \Phi \wedge \Phi \mid \neg\Phi | \mathbb{P}_{\bowtie p}[\Psi] \quad (1)$$
$$\Psi ::= \mathbf{X}\Phi \mid \Phi\ \mathbf{U}^{\leq k}\ \Phi \mid \Phi\ \mathbf{U}\ \Phi \quad (2)$$

where :

- c is an atomic proposition ;
- $k \in \mathbb{N}, p \in [0,1], \bowtie \in \{\leq, \leq, \geq, \geq, =\}$.

Figure 3. The syntax of the logic PCTL

$$\begin{array}{llcll}
s & \models_{Pol} & \textbf{true} & & \text{always} \\
s & \models_{Pol} & c & \Leftrightarrow & c \in L(s) \\
s & \models_{Pol} & \Phi_1 \wedge \Phi_2 & \Leftrightarrow & s \models_{Pol} \Phi_1 \wedge s \models_{Pol} \Phi_2 \\
s & \models_{Pol} & \neg\phi & \Leftrightarrow & \not\models_{Adv} \Phi \\
s & \models_{Pol} & \mathbb{P}_{\bowtie p}[\Psi] & \Leftrightarrow & P^{\sigma}_{M,s}(\{\pi \in Path^{\mathcal{M}}, \pi \models_{Pol} \\
 & & & & \Psi\}) \bowtie p, \forall\sigma \in Pol
\end{array}$$

where, for any $\pi \in Path^{\mathcal{M}}$:

$$\begin{array}{llcll}
\pi & \models_{Pol} & X\phi & \Leftrightarrow & \pi(1) \models_{Pol} \Phi \\
\pi & \models_{Pol} & \Phi_1\ \mathbf{U}^{\leq k}\ \Phi_2 & \Leftrightarrow & \exists i \leq n,\ \pi(i) \models_{Pol} \Phi_2 \\
 & & & & \text{and } \forall j < i,\ \pi(j) \models_{Pol} \Phi 1 \\
\pi & \models_{Pol} & \Phi_1\ \mathbf{U}\ \Phi_2 & \Leftrightarrow & \exists k > 0,\ \pi \models_{Pol} \Phi_1\ \mathbf{U}^{\leq k}\ \Phi_2.
\end{array}$$

with :

- $P^{\sigma}_{M,s}A$ is the probabilistic measure of the set A on the MDP M from the state s with policy σ ;
- $\pi(k)$ is the k^{th} state of the path π (if π starts from s, then $\pi(0) = s$) ;
- Pol is the set of policies we consider.

Figure 4. The semantics of the PCTL logic

need to be able to express temporal properties (like the fact that an event may or may not happen). Moreover, it has to allow probabilistic properties. That is why we use the Probabilistic Computational Tree Logic (PCTL in short) [8], a probabilistic variant of the branching-time temporal logic CTL (Computational Tree-Logic) [9].

The syntax of the PCTL logic is given in figure 3. We distinguish the state formulas Φ (line (1)) and the path formulas Ψ (line (2)). The PCTL formulas we will consider will be state formulas.

PCTL formulas do not express nondeterminism. That is why, when they are evaluated on an MDP, we use a quantification over all policies (that is, over all DTMC that can obtained from that MDP), that erases nondeterminism.

Intuitively, a state s satisfies the probabilistic path operator $\mathbb{P}_{\bowtie p}[\Psi]$ ($\bowtie \in \{<, >, \leq, \geq\}$) if, under all policies, the probability x of taking a path from s that satisfies Ψ is such that $x \bowtie p$. The temporal operators are expressed in path formulas. Intuitively, $\mathbf{X}\Phi$ will be true if Φ is true in the next state. $\Phi 1\ \mathbf{U}\ \Phi 2$ is true if eventually $\Phi 2$ will be true and until then $\Phi 1$ is true. In the same way, $\Phi 1\ \mathbf{U}^{\leq k}\ \Phi 2$ is true if $\Phi 2$ is true within k time-steps and until then $\Phi 1$ is true. The formal semantics is given at figure 4.

A usual shortcut is the operator $F\ \Phi$ witch stands for $true\ \mathbb{U}\ \Phi$. Intuitively, this operator states that eventually Φ will be true. $F^{\leq k}\ \Phi$ can be defined in the same way.

Let us look at some example of PCTL formulas. The safety property "two trains will not collide" is equivalent to "the probability of two trains colliding is equal to 0". Let Φ be the (state) formula expressing that two trains are colliding, then the PCTL formula expressing the above property is $P_{\leq 0}[F\ \Phi]$. Here is another example. The property "a train recovers from its delay within 10 steps with a high probability" could be described by the PCTL formula $P_{\geq 0.9}[F^{\leq 10}\ \Phi]$ if Φ expresses that a train has no delay.

There is an other way to express PCTL formulas in MDPs. We can define a minimum and maximum probability on MDP satisfying a path formula Ψ.

Definition 4. *Let $\mathcal{M}$ be an MDP and Ψ a path formula. We define:*

$$P^{\mathcal{M}}_{min}(\Psi) = min_{\sigma \in Policy} P^{\mathcal{M}^{\sigma}}(\Psi)$$
$$P^{\mathcal{M}}_{max}(\Psi) = max_{\sigma \in Policy} P^{\mathcal{M}^{\sigma}}(\Psi)$$

where Policy is the set of all policies of the MDP $\mathcal{M}$.

We implemented the MDP of the figure 2 in PRISM [10] to compute the minimum and maximum probability to reach state S2 in less that 10 steps starting from state s_0:

$$P_{min}(F^{\leq 10}(state = s2)) = 0$$
$$P_{max}(F^{\leq 10}(state = s2)) \simeq 0.989$$

We can then express PCTL formulas (that is, state formulas) by replacing the operator $\mathbb{P}_{\bowtie p}[\Psi]$ by $\mathbb{P}_{min\ \bowtie p}[\Psi]$ or $\mathbb{P}_{max\ \bowtie p}[\Psi]$. The semantics is analogous except that we do not evaluate $\mathbb{P}$ but $\mathbb{P}_{min}$ or $\mathbb{P}_{max}$. There is no need for quantification over policies anymore since this quantification already occurs in the definition of P_{min} and P_{max}.

From the above results, we can deduce that, for example, the formula $P_{min \geq 0.8}(F^{\leq 10}(state = s2))$ holds for the MDP whereas $P_{max \leq 0.9}(F^{\leq 10}(state = s2))$ does not.

C. PRISM: a probabilistic model checker

To verify PCTL properties on MDP or DTMC models, one can use probabilistic model checkers. The one we use is PRISM[1] [10] which is a state-of-the-art tool in this field, and is also quite efficient at verifying probabilistic properties on MDP and DTMC models. PRISM is easy to use with its graphical user interface which offers multiple functionalities, like verifying that a PCTL formula holds on an MDP, debug a model or export the model behavior in different formats. It takes two inputs: the model and the properties (expressed as PCTL formulas) we want to verify on it. Then PRISM returns the result of the evaluation of these properties on this model.

PRISM has its own language to describe a model. It is a symbolic language: it describes a model without giving every states and transitions (as an explicit language would do it). The model is separated into different processes. In our case, processes can be the different trains in the network. These processes can be synchronized and run in parallel: at each time step, every trains can move.

The PRISM language uses modules to describe processes. A module contains two parts: the definition of state variables (with its definition's domain), and several actions. A state variable represents a characteristic of a process (which can be boolean or integer for example). For instance, we define a variable `stationi` to define the position (as an integer) of the train i. An action has a guard (that is a boolean expression) which specifies if the action can be taken. In our case, a guard can state if a train has waited long enough

[1] http://www.prismmodelchecker.org/

at a station. If an action is taken, the evolution of the variables is probabilistic (the sum of all probabilities occurring on a given action must be equal to one). Actions can also be named. In that case, every action with the same name in different modules are synchronized (if one is taken, every possible one in the other modules is taken). The action step, in the Figure 5, describes the movement of train 1. If the action is allowed, (that is, if station2 $\geq$ station1 + 1), then the number of stations between train 1 and train 2 is greater than one. In that case, train 1 will move forward with a probability equal to 0.8 and will stay at the same station with a probability equal to 0.2 (we have $0.8 + 0.2 = 1$). The action change, in the Figure 5 changes the value of the boolean variable switch. It can be taken at every step (since the guard is *true*). In each module, if there is more than one actions available, a nondeterministic choice is made to decide which one to take.

The logic to express properties in PRISM is PCTL. These properties are defined for a model while using the variables used to define that model. However, to make easier the writing of properties, PRISM uses labels. A label is a boolean expression that acts as a sub-formula. If a label is defined, it can be used directly to write properties, but it can not be used to describe the model. Moreover, the evaluation of a property in a given model can yield either a Boolean response or a probability. The Figure 6 illustrates theses possibilities. The label isDelays01 stands for: "train 1 has some delay". The first property requests PRISM to compute the maximum probability that if train 1 is late, it will eventually recover from this delay. The second property requests from PRISM a Boolean response that answers the question: is the previous probability above 0.9 ?.

PRISM can verify that properties hold in a model. With the model description, PRISM computes the reachable states and every transitions available is these states. A state is described by the value that every variables are affected to in every module. The transitions available in a state are given by all the actions whose guard is allowed with the values of the variables in that state. Once the model is built, properties can be evaluated on it. However, finding a deadlock or, more generally, writing properties to verify the soundness of the model is not easy. Therefore, a way to check the soundness of the model without writing PCTL formulas is to use the PRISM simulator. It allows us to see the evolution of the system along some specific paths. We can then see the value of every variable in the states we pass by with this path. It can be an effective way to find a bug in the model, for instance a deadlock. However, the simulator can not give any proof that a property does or does not hold in a model.

Moreover, when the model is too big, PRISM can not build the model, and the properties can not be evaluated on the model. In that case, we can use the statistical model checking that PRISM offers. This method consists in building a lot of possible paths (as much as requested to ensure a given confidence interval) and computing the ratio of paths satisfying a desired property (which will be a PCTL path formula) among all explored paths. This yields an approximation of the probability that this property holds in the model. The main advantage of this method is that PRISM does not need to effectively build the model to compute these paths. PRISM can use several methods to compute these paths. In each method, we can control different parameters like the length of the paths (the number of steps) or the accuracy of the approximation.

D. Abstraction

PRISM can not work on a very large model. So, should the size of the model (number of states and transitions) be overwhelming for PRISM and we want to prove a property (not approximate a probability), we shall need to reduce its size. A well-known method to achieve this is to design an abstraction of our model. Intuitively, abstracting an MDP is building a new MDP based on the previous MDP deprived of some irrelevant informations (regarding some specific property of interest). Abstracting an MDP consists in building a new MDP in which some states are merged in order to reduce their number. The transitions in the new MDP are then derived from the initial ones.

There are several ways to abstract an MDP. We are only going to talk about the quotient of an MDP [12] since it is the method we will use. Given an MDP $\mathcal{M} = (S, Act, \mathbf{P}, s_{init}, AP, L)$ and a partition $\mathcal{Q} = q_1, ..., q_n$ of the state space S that respects the labeling, that is: for all q_i and $s, s' \in q_i$, $L(s) = L(s')$, an abstract MDP consists in aggregating every states in the same set q_i into one state. The difficulty lies in the definition of the behavior of the abstract MDP (in terms of transition). An abstract state q_i must have the behavior of the states that it has merged. In other words, if we have $s \in q_i$, $s' \in q_j$ and an action $\alpha \in Act$ such that $\mathbf{P}(s, \alpha, s') > 0$, then there must be an action α' that 'links' abstract states q_i and q_j. Formally we have the following definition [12]:

Definition 5 (Abstract MDP)**.** *Let an MDP* $\mathcal{M} = (S, Act, \boldsymbol{P}, s_{init}, AP, L)$ *and a partition* $\mathcal{Q}$ *of the state space* S *that respects the labeling. The quotient of* $\mathcal{M}$ *by* $\mathcal{Q}$ *is given by the abstract MDP* $\mathcal{M}/\mathcal{Q} = (\mathcal{Q}, Act', \boldsymbol{P}/\mathcal{Q}, q_{init}, AP, L/\mathcal{Q})$ *where*

- Act' *is the set of action to use in the abstract MDP, after renaming the actions in* Act*;*
- $\boldsymbol{P}/\mathcal{Q}$ *is a transition probability matrix such that* $(\boldsymbol{P}/\mathcal{Q})(q, \alpha', q') = \sum_{s \in q'} \boldsymbol{P}(s, \alpha, s')$ *(where* α' *is a renaming of* α*);*
- q_{init} *is the set of initial state such that* $q \in q_{init}$ *if there is* $s \in q$ *such that* $s \in s_{init}$*;*
- $L/\mathcal{Q}$ *is the labeling function such that* $(L/\mathcal{Q})(q) = L(s)$ *for all* $s \in q$*.*

Note that the labeling is well-defined, since $\mathcal{Q}$ respects labeling. Moreover, we need to rename the new action set since every name of actions in the abstract model should be annotated with the an indicator of the actual state its depicting.

The abstraction method is used to reduce the size of a model while keeping its behavior. Therefore, the abstract model is less precise in the description of that behavior. In fact, generally, in an abstract MDP, the probability of going from one state to another is defined by an abstract probability, that is a set of possible probabilities (since an abstract state corresponds to several actual states). That is why abstracting a model yields a new source of nondeterminism in the model. Therefore there are two sources of nondeterminism in an abstract MDP:

- nondeterminism due to choice of actions, as in the original MDP;
- non-determinism due to uncerttainty on the actual state inside an abstract one.

It induces, in the abstract MDP, an over approximation of probabilities. That can be seen in the following theorem [12].

Theorem 1. *Let* $\mathcal{M}$ *be an MDP,* $\mathcal{Q}$ *be a partition of its states space and* Ψ *a PCTL path formula, then*

$$P_{min}^{\mathcal{M}/\mathcal{Q}}(\Psi) \leq P_{min}^{\mathcal{M}}(\Psi) \leq P_{max}^{\mathcal{M}}(\Psi) \leq P_{max}^{\mathcal{M}/\mathcal{Q}}(\Psi)$$

This theorem states that the probability to satisfy a PCTL formula

```
   [step]      station2 ≥ station1 + 1    -> 0.8 : (station1' = station1 + 1) + 0.2 : (station1' = station1);
  [change]     true                       -> 1: (switch' = !switch);
   Action            Guard                        Modification of the variable station1
```

Figure 5. Commands expressed in the PRISM language

```
Pmax=? [("isDelay01" U !"isDelay01")]
Pmax>=0.9 [("isDelay01" U<=20 !"isDelay01")]
```

Figure 6. PCTL formula described with PRISM language

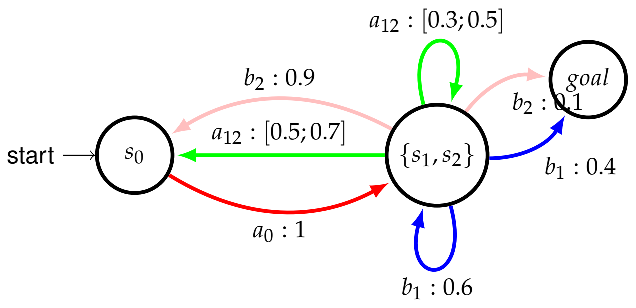

Figure 7. Abstraction of the MDP from figure figure 2 wuth the partition $\mathcal{Q} = \{\{s_0\};\{s_1,s_2\};\{goal\}\}$

Ψ in an MDP lies between the probabilities to satisfy it in its abstraction.

III. First approach

A. Our Model

On a subway rail system, a train faces random events and may be delayed. These delays are not predictable, hence we need to use probabilities to correctly model the system. Moreover, trains can adjust their dwell time in station to recover from delays. This is modeled by nondeterminism. That is why we use Markov Decision Process to represent the rail system for it is both probabilistic and nondeterministic.

MDP is a discrete model and a subway rail system is obviously continuous. Therefore we need to discretize time and space. In the real system, train moves are continuous. In the MDP, trains move forward step by step, with every step corresponding to a fixed amount of time. We discretize the distance between two stations by adding intermediate points between these stations. The distance between two points is also fixed. In this model, the speed is inversely proportional to the time needed to travel between two stations.

As mentioned in the introduction, we want ot study the efficiency of regulation policies. That is why we made the hypothesis that the model we are depicting ensures the safety properties (that is, there is no collision between trains). For our model to be correct, it has to respect this hypothesis independently of any regulation policy. In terms of PCTL formula, an MDP $\mathcal{M}$ that models a subway rail system needs to ensure that the formula $P_{max\leq 0}(F\ collision)$ holds.

We begin by modeling a single rail line. More specifically we are going to work on the subway line of Glasgow, that can be seen in figure 8. This line is a ring without intersection. Larger networks might contain intersections, which are more difficult to handle because we have to determine priorities on trains that need to enter the intersection. On this ring, trains always move in the same direction.

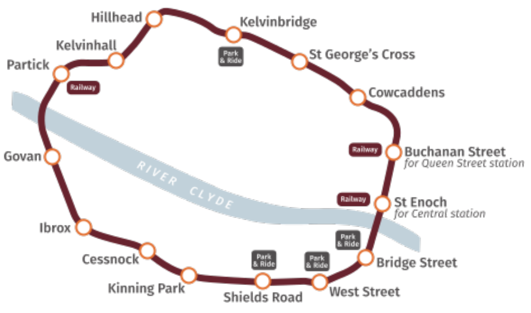

Figure 8. A schema of the subway line of Glasgow

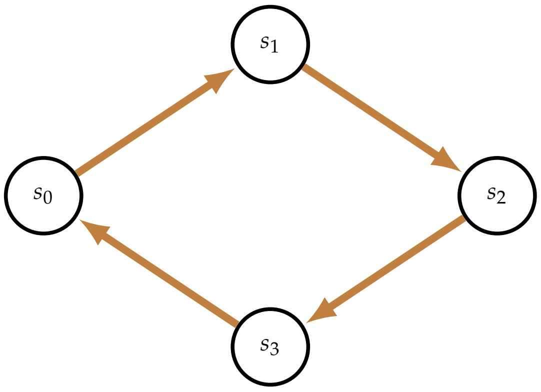

Figure 9. The simplified model we first want to study (with four stations)

As mentioned earlier, we introduce intermediate points between stations to take the discretization of space into account. We call location either a point or a station. In this way, the distance between two stations is depicted by the number of intermediate points between two consecutive stations, given that the distance Δd between two locations is constant. At each step, trains may move forward from a location to another. The duration Δt of the steps fixes our discretization of time. The smaller the distance (and the duration) the more accurate the discretization.

Let us consider the MDP of Figure 9 that is an abstraction of the subway system of Figure 8 where we have only four stations. The section between stations is depicted by the figure 10.

At each location, there are three possible actions. Action a yields the green transitions, whereas action b yields the blue transitions and action c the red ones. Note that all three actions are not necessarily always allowed. Indeed, this model needs to ensure safety measures. For example, actions a and b are forbidden at each location where there is a train in the next location. This restriction should ensure that trains do not collide.

In this model it is easy to associate a speed to each action. Trains can only stay at their current locations or go to the next one at each step. By looking at the probability to go forward at each step, we note that action b corresponds to a lower speed than action a and

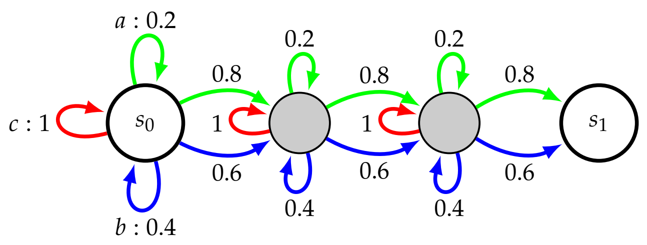

Figure 10. The modelization between two stations with two intermediate points

action c forces a train to stop at a location.

To be more precise, we can compute the speed induced by choosing each action. For example, let us consider action a. The probability of going to the next location in one step is equal to $p_a = 0.8$. It is straightforward that the probability of going to the next location at step $k \geq 1$ is equal to $(1 - p_a)^{k-1} \times p_a$. Hence, that probability follows a geometric law. Therefore, the expected value of such a law is equal to $E_a = \frac{1}{p_a} = \frac{1}{0.8} = \frac{10}{8} = 1.25$. That means that, on average, it will take E_a steps to go from a location to the next one. So, the speed v_a corresponding to action a is equal to :

$$v_a = \frac{\text{distance between two succesives locations}}{\text{average time needed to go from a location to the next one}}$$
$$= \frac{\Delta d}{E_a \times \Delta t} = \frac{\Delta d}{1/p_a \times \Delta t} = p_a \times \frac{\Delta d}{\Delta t} = \frac{4 \times \Delta d}{5 \times \Delta t}$$

In the same way, we can compute the average speed obtained by choosing action b. That is:

$$v_b = p_b \times \frac{\Delta d}{\Delta t} = \frac{3 \times \Delta d}{5 \times \Delta t}$$

Finally, we have that the speed for action c is $v_c = 0$. These results show that we can model trains traveling at different speeds with probabilities in our discrete MDP model.

B. Choosing parameters

The next step is to choose the parameters of the model to better picture the real system, in our case, the Glasgow subway. We will then consider a system with $nb_{station} = 15$ stations. To simplify the model, we will assume that the stations are equally spaced. The parameters we still have to determine are:

- the time discretization step: Δt;
- the space discretization step: Δd;
- the probabilities of going forward: p_a, p_b;
- the number of intermediate steps between two stations: k;
- the number of trains: nb_{train}.

We collected information from existing Metro lines, and from Glasgow in particular[13]. The speed of train in sybway systems lies between 30 and 40 $km.h^{-1}$. And the average time waited in a station is around 30s. Specifically to the Glasgow subway, we know that a complete circuits is 10.5 km long (it is actually 10.4 km, we will assume it is 10.5 km since it is divisible by 15) and lasts 24 min.

With this data, we can infer that the effective time spent on the rail to run through the circuits is equal to $t_{tot} = 24 \times 60 - 15 \times 30 = 990s$. That corresponds to an actual speed of $v_{av} = 10500/990 \approx 10.6\ m.s^{-1} \approx 38.2\ km.h^{-1}$ which lies between 30 and 40 $km.h^{-1}$.

Let us first focus on the probabilities p_a and p_b. In our model, we want the speed induced by action a to be the usual speed of trains, and the speed corresponding to action b to be a lower speed occurring when trains slow down for safety reasons (if the next train is too close).

The data we have from the actual system give us relations between our parameters. The distance between two stations is equal to $d_s = k \times \Delta d\ (= d_{tot}/nb_{station} = 10500/15 = 700\ m)$, and the average speed is equal to $v_{av} = p_a \times \Delta d / \Delta t$ (as already seen before). We can deduce the average time taken between two stations t_s which is $t_s = d_s/v_{av} = k/p_a \times \Delta t$. Therefore, we have the relation $k \times \Delta t = p_a \times t_s$ while t_s is fixed in the system: $t_s = t_{tot}/nb_{station} = 990/15 = 66\ s$. That is, we have the relation:

$$k \times \Delta t = p_a \times 66\ s$$

Once p_a is fixed, $k \times \Delta t$ is a constant. To reduce the size of the model, we want to minimize k since it defines the number of intermediate steps between two stations. Moreover, the smaller Δt, the higher the accuracy of the discretization. In fact, we want to minimize both k and Δt. That is why, we want p_a to be as small as possible.

We wanted to choose p_a so that it fits some typical data from an actual subway line. We did not have data from Glasgow, but we had some from the subway of Santiago. A typical distribution is given in figure 18 in appendix A. The blue histogram depicts the actual distribution to go from one station to the next one in the Santiago subway. We can see, as expected, that is more likely to take some delay than to be in advance compared to the nominal time (the mean of the distribution). On this histogram, four curves are drawn which correspond to the probability distribution for the time needed to go from one station to the next one in our model. These curves represent the sum of k independent geometric law of the same parameter p, with the same mean E as the blue distribution. The corresponding probability law can be found in appendix. *Dist* corresponds to the sum of the difference between the distribution and the curves squared.

Our goal is to determine which one fits the best the blue histogram. However, it is not easy to choose which one is the more appropriate. Indeed, the shape of a curve is better (i.e. is closer to the actual distribution) when p is higher since it depicts well that it is very difficult to take some advance but it is possible to have some delay. However, a higher p also means a higher *Dist*, which is suppose to represent how well a curve fits the distribution. Moreover, we have to keep in mind that we want to minimize p_a. We decided to take $p_a = 0.8$. That choice is quite arbitrary but no choice seems obviously better than this one. Then, we choose $p_b = 0.6$ so that, if p_a corresponds to a speed around 40 $km.h^{-1}$ (the maximum usual speed of trains), then the speed associated to p_b is around 30 $km.h^{-1}$ (the minimum usual speed of trains).

Then, once k is fixed, Δt can immediately be deduced, as Δd since we have: $k \times \Delta t = p_a \times 66\ s = 52.8\ s$ and $k \times \Delta d = 700\ m$ (see above). Then, we consider two sets of parameters:

- $k = 5$
- $\Delta t \simeq 10\ s$
- $\Delta d = 140\ m$

- $k = 10$
- $\Delta t \simeq 5\ s$
- $\Delta d = 70\ m$

The first set favors the size of the model ($k = 5$), the second one, on the other hand, corresponds to a larger model ($k = 10$) but with a increased accuracy ($\Delta t \simeq 5s$). It has to be noted that it is not quite possible to increase k above 10 since it would imply that the space discretization step could be smaller than the size of a train (which is not possible in the current model since a train can only be in one location at a time).

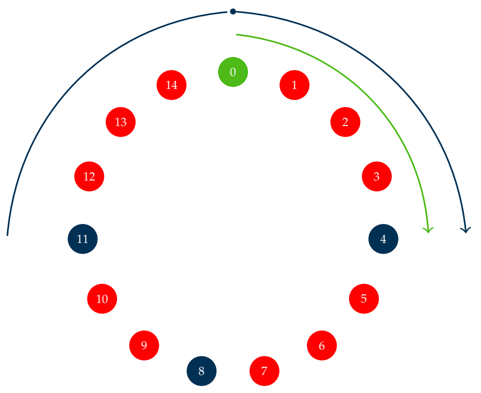

Figure 11. A schema that illustrates our definition of α

Finally, we have to choose the number of trains of interest. From the data extracted from the Glasgow subway website[13], at off-peak time, there is a train every six minutes, with a complete circuit lasting 24 min, this corresponds to $nb_{train} = 24/6 = 4$. And at peak-time, there is a train every 4 minutes, which corresponds to $nb_{train} = 24/4 = 6$. Therefore, 4 and 6 are the number of trains of interest for us.

C. Our regulation policy

We have defined the MDP that will represent the Glasgow subway. We can now design a regulation policy on that MDP. A regulation policy will consist, in our MDP, in resolving the nondeterminism, that is deciding which action to take when a choice can be made. The only choice that is possible in our MDP is the dwell time in station. The nominal time waited in station is 30 s. However, we allow a train to stay from 20 s up to 40 s. It has to be noted that the more accurate our discretization is, the more choices there are in station. Indeed, with $\Delta t \simeq 10$ s there are three possibilities: staying $2, 3$ or $4 \times \Delta t$. However, with $\Delta t \simeq 5$ s, we have five possibilities: staying $4, 5, 6, 7$ or $8 \times \Delta t$.

A regulation policy aims at avoiding delays or recovering from them. Thereby, we first need to formally define what we call a delay. The definition we chose for a delay comes from the quality of service that is expected from subway networks. Indeed, it is not required from train to adhere to a specific timetable, but rather to provide service with a given frequency. That is why we define a value α that measures whether time intervals between trains are equilibrate or not. Let us consider the schema 11. Every circle represents a station. The red circle are empty stations. There is a train in each blue circle. And the train of interest for us is in the green circle. We want to measure how that train is located compared to the next and the previous ones. We define: $\alpha = \dfrac{d(current, next)}{d(previous, current) + d(current, next)}$ with $d(,)$ measuring the number of points between two trains. The numerator corresponds to the small arrow and the denominator to the long one, in the schema. It has to be noted that α cannot be equal to 0 nor 1 since two different trains cannot be in the same location. Moreover, α is defined for every train in the system.

We can now define what a delay is (a more suitable denomination would an unbalance) for a train. Ideally, α is equal to 0.5 (it is the case in the schema of figure 11) which means that the current train of interest is perfectly in the middle of the space delimited by its preceding train and its following one. However, this definition is too restrictive in practice, since there would be at most one location allowed from a balanced train (there could be none if the distance between the previous and the next train is odd). Henceforth, a train is unbalanced if $\alpha \notin [0.4, 0.6]$. We say that the system is balanced if every train is balanced.

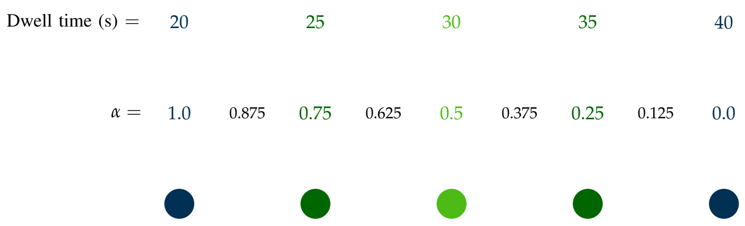

Figure 12. The dwell time in station in function of α

We can now define our regulation policy. We decided to define it as a function of α. The idea is very simple: the time waited will linearly depend on α. If α is close to 1, the train leaves early, if it is close to 0, it has to dwell in station for a longer duration than the nominal dwell time. The schema 12 illustrates this idea. Due to the discretization, the dwell time in station depends on the interval to which α belongs and not the exact value of α. For instance, with $\Delta t \simeq 5$ s, the dwell time was decided as follows:

$$\text{Dwell time}(\alpha) = \begin{cases} 4 \times \Delta t & \text{if } \alpha \in [0.875, 1] \\ 5 \times \Delta t & \text{if } \alpha \in [0.625, 875] \\ 6 \times \Delta t & \text{if } \alpha \in [0.375, 0.625] \\ 7 \times \Delta t & \text{if } \alpha \in [0.125, 375] \\ 8 \times \Delta t & \text{if } \alpha \in [0, 0.125] \end{cases} \tag{3}$$

We use the same regulation policy when $\Delta t \simeq 10$ s. That is the regulation policy we implemented.

D. Desired properties

Now that our model is entirely defined, as well as our definition of delay and our regulation policy we want to check some properties on the model. First, we have to express in PCTL the property that states that there cannot be any collision. We have already mentioned that the PCTL formula that has to hold in our model is: $P_{max \leq 0}(F\ "collision")$. We now have to define $"collision"$. Since the position of train i is described by the variable $station_i$, then

$$"collision" = \bigwedge_{i=1}^{n} \bigwedge_{j=1, j\neq i}^{n} \left(station_i \neq station_j\right)$$

with n being the number of trains in the system.

We can now look at whether the system is balanced. First, for a train i, let us define $\alpha_i = station_{i+1} - station_i (mod\ l) / station_{i+1} - station_{i-1} (mod\ l)$, l being the total number of locations in the model. Then, it is straightforward to express the formula which states that our system is balanced:

$$"balanced" = \bigwedge_{i=1}^{n} (0.4 <= \alpha_i <= 0.6)$$

Therefore, the formulas we want to evaluate on our model are:

$$recovering = P_{min=?}(F_{<=q} "balanced")$$

$$taking = P_{max=?}(F_{<=q} \neg "balanced")$$

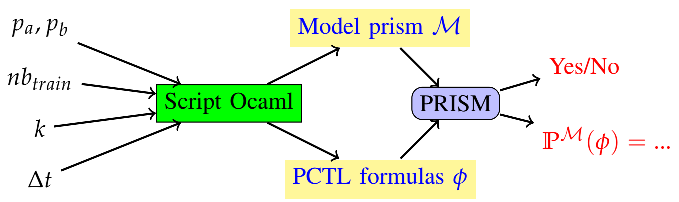

Figure 13. A graphical sum up of what we did

with q being a given number of steps. *recovering* asks the minimum probability that the system is, at some points, balanced within q steps. It has to be evaluated from an unbalanced initial configuration. *taking* asks the maximum probability that the system is, at some point, unbalanced within q steps. It has to be evaluated from a balanced initial configuration.

E. Implementation in PRISM

Now that everything is theoretically defined, we can effectively implement our model in PRISM, with the properties we want to evaluate on it. We implemented a script in Ocaml that takes as input parameters (for instance p_a, p_b, nb_{train}) and outputs the definition of the model in the PRISM language and the properties of interests for us in two different files. Then, PRISM takes as input these two files and gives us the evaluation of the properties on the model. This is summed up in figure 13.

However, with the smaller model that was of interest for us (with $k = 5$ and $nb_{train} = 4$), PRISM was not able to build the model. Indeed, the number of configurations to consider for subway systems reached the limits of model checking. In such situations, it is hence necessary to use statistical model checking, or reduce the size of the explored state space, either by simplifying the initial model, or through abstraction techniques.

IV. Second approch

A. Abstraction

We cannot work on this model since it is too big for PRISM to build it. So, we need to reduce its size. As, we saw in the subsection II-D, we may use an abstraction to effectively reduce the size of the model.

Abstracting a model consists in building a quotient of a model. We can notice that, our definition of α (on which entirely depends both the definition of a balanced system and our regulation policy) only relies on the distance between trains, not their exact location. In fact, we do not need to define the position of trains exactly in the system but only relatively to the other trains.

We can define an equivalence relation between different situations of the system to characterize this abstraction. Two states of our system are said equivalent if and only if for all pair of trains, the distance (as a number of locations) between them is the same in both situations (all other variables, in particular the time waited in station, being unchanged). In fact, this is equivalent to requiring that a configuration of trains can be obtained from the other by a rotation (that is, every train is moved in the same direction, of the same number of locations). That can be seen in appendix A. It has to be noted that the number of locations of the rotation has to be a multiple of the number of intermediate steps between two stations (since the fact that a train is in station or not should be preserved).

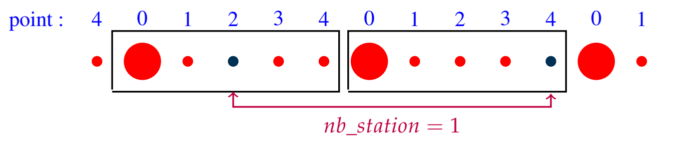

Figure 14. Definition of the new variables for our abstraction (with $k = 5$).

Model Number of trains	Before abstraction	After abstraction
Three trains	2.1×10^8 states 3.4×10^9 transitions	3.5×10^5 states 8.3×10^5 transitions
Four trains	Not built in PRISM	2.0×10^7 states 5.7×10^7 transitions

Figure 15. Size of the model in terms of number of states and transitions (in both cases, $k = 5$)

We can define formally an equivalence relation $\mathcal{R}$. Let s, s' be two states with $station_i$ and $station'_i$ representing the location of train i respectively in s and s'. Then: $s\mathcal{R}s'$ if and only if there exists $a \in \mathbb{N}$ such that $\forall i \in [\![1, nb_{train}]\!]$, $station_i = station'_i + a \times k\ [q]$ with k defined in section III-B, q being the total number of locations in the system and every other variable is unchanged. With that relation, we can define the partition $\mathcal{Q}$ as all the classes of equivalence of the relation $\mathcal{R}$. We can then obtain a new model by using the method described in subsection II-D.

This abstraction gives us a new MDP. The position of trains are now described relatively to the other trains. It is depicted by two variables (that is shown in figure 14):

- *point*: distance between a train and its previous station;
- *nb_station*: number of stations between a train and its successor.

These variables are enough to compute the distance between two consecutive trains and, consequently, between any two trains. Indeed, given a train i, $d(train_i, train_{i+1}) = k \times nb_{station\ i} + point_{i+1} - point_i$. It means that, these new variables are enough to define α for every train, and therefore our regulation policy and the properties we want to check.

We analyze the impact of our abstraction on the size of the model. A quick comparison is shown in figure 15. First, with three trains, PRISM can build both models, but the size of the model is much smaller with our abstraction, since there are approximately 10^3 less states and 10^4 less transitions. But, more importantly, PRISM is now able to build the model with four trains, which was not possible without our abstraction. It means that we can evaluate our regulation policy for the off-peak hours in the Glasgow subway.

B. Soundness of the model

We now need to check that our model satisfies our initial hypothesis assuming that the system is safe. This property cannot be proved with statistical model checking (since it studies only a finite number of paths), that is why we need PRISM to effectively build the complete state space of the MDP representing the behavior of a subway system. With our abstraction, PRISM can build the model with $k = 5$ and $nb_{train} = 4$. PRISM has been able to prove that our model ensures the safety property.

However, the state space cannot be built as soon as $nb_{train} = 6$ or $k = 10$. Therefore, we need a new abstraction. We designed a new abstraction specifically to prove the safety property. We designed a new MDP whose behavior encompasses the behavior of the previous one. In the new MDP, every transition is nondeterministic such that, if a transition is available (and weighted by probabilities) in the initial MDP, then it is also available as a nondeterministic choice in our new MDP. Moreover, there is no more a distinction between stations and intermediate points since we do not consider dwell time anymore. The only restriction on the transition existed in the previous MDP and it is that a train cannot go forward if the next train is at the next location.

By definition, any valuation of the variables that was reachable in the previous MDP is also reachable in the new MDP. In particular, that means that if the new MDP ensures the safety property, then so does the previous MDP, which is our model of interest.

With that new abstraction, PRISM was able to build the model for $k = 5, 10$ and $nb_{station} = 4, 6$, and prove that it ensures the safety property. It shows that our model respects our initial hypothesis.

C. Statistical Model Checking

We finally can evaluate the efficiency of our regulation policy. There are several ways to evaluate it. We first have to decide which initial position we consider. Considering the computing time necessary to correctly evaluate a property from an initial position, we must make a choice of a small number of initial situation we will consider. We defined around 15 initial position of interest that were representative of different situations that could occur in the system.

We present here the result from a specific position. We can draw the same conclusion from the other initial positions. The initial position we consider is the position where every trains is in a single file. We chose to show this one since it is a situation that can occur on a real system and it is interesting to see how the system recovers from a totally unbalanced situation.

We can see in appendix A the three graphics drawn presenting three different situations. The first one is drawn with $k = 5$ and $nb_{train} = 4$. In that situation, PRISM can build the model and it allows us to use exact model checking. However, we only used it to draw the green curve since the other two were far too long to draw (more than 4 hours to compute the first point of the first curve0). All curves are drawn by evaluating the property *recovering* from section III-D. That is, we see the evolution of $P_{min=?}(F_{<=q}"balanced")$ (the y-axis) as q grows (the x-axis, converted in minutes). The green curve shows the optimal solution, with a perfect regulation policy. However, such a perfect regulation policy cannot be used in practice since PRISM evaluates a response for every valuation of the variables. But, it gives us optimal regulation that we can try to approach. We drew two line to associate a time to two given probabilities (0.5 and 0.8) that seemed of interest. We can see the optimal solution is far better than ours (the red curve). The blue curve correspond to a situation without any kind of regulation policy (trains always leave after the nominal dwell time in station). In this situation, it seems very complicated to recover from such a initial situation. On the other hand, our regulation policy seems able to recover from such a unbalanced situation.

The second graphic shows the result obtained when $k = 10$. The exact model checking could not be used in this case since the model could not be built with PRISM. The other two curves seem to have the same shape than previously. However, our regulation policy seems a little more effective than previously since the time to reach $p = 0.5$ and $p = 0.8$ is smaller than previously (however these results are undercut by the uncertainty on the curves, one can wonder about the significance of such a little difference).

Finally, the third graphic shows the result obtained when $k = 10, nb_{train} = 6$. We can compare this result with the previous one (with $k = 5$). It seems that with 6 trains, recovering is faster (one achieves a higher probability to recover for a given q) than with 4 trains, which seems normal.

These results shows that our regulation policy is far better than no regulation at all (which is the least we can expect). However, we can see how far we are from the optimal regulation.

V. Conclusion

Our goal was to evaluate the efficiency of regulation policies in subway systems. We have designed a model of the Glasgow subway and implemented it on PRISM. The model we used is Markov Decision Process which allows to combine probabilities and nondeterminism. Thanks to some abstraction, we were able to prove the soundness of our model. Then, we have effectively evaluated our regulation policy. With this regulation, trains seem able to recover from delays in reasonable time.

We used both exact and statistical model checking throughout this project. However, we did not use model checking at its full potential. For instance, we could have designed an initial situation and specified that in the first hour a train will take serious delay without having to specify when, thanks to the nondeterminism, and observe how the system would have evolved. This is a kind of study that is not possible with simulation.

An interesting way to evaluate our regulation policy would have been to design another regulation policy and compare the two (it has not been done because of a lack of time), since the optimal regulation policy we considered cannot be realized in practice.

A lot of choices have been made during this project and they could be reconsidered. Our definition of delay could be strengthen (for example, a train has some delay if $\alpha \notin [0.45, 0.55]$) or soften ($\alpha \notin [0.3, 0.7]$). The impact it would have on the curves we have drawn would be interesting to study. Moreover, the definition of α could be changed. For example, it could evolve according to the number of trains in the system.

The model itself could also be reconsidered. The choice of p_a, p_b or our way of depicting the speed of trains could be different. Finally, we could consider a more complex topology of subway system.

Acknowledgment

We want to thanks our supervisors Nathalie Bertrand, Loïc Hélouët and Ocan Sankur for their time and their help all along this year.

Appendix

Equivalence by rotation

The two figures 16 and 17 are equivalent in terms of balance of the system.

Justifying probabilities

The repartition can be seen at figure 18. Each curve corresponds to the sum of k independent geometric law of parameter p. The probability law of these curves is:

$$\mathbb{P}(S_k = n) = \begin{cases} 0 & \text{if } n < k \\ \binom{n-1}{k-1} \times (1-p)^{n-k} \times p^k & \text{otherwise} \end{cases} \quad (4)$$

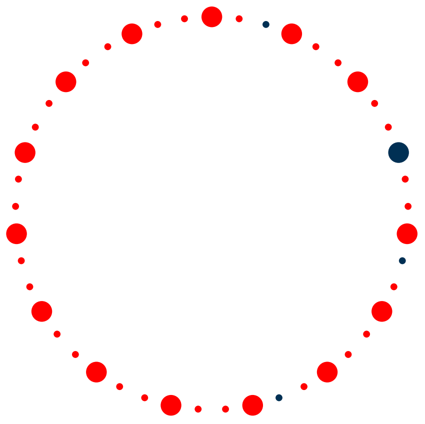

Figure 16. A first configuration with $k = 3$ (subways are where the point is blue)

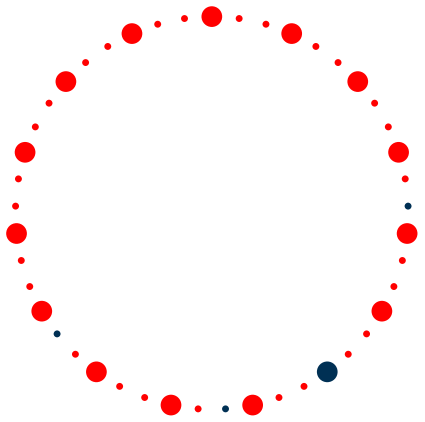

Figure 17. A second configuration with $k = 3$ equivalent to the first one (subways are where the point is blue)

Efficiency of the regulation policy

The graphic that were drawn from the initial position where every subway are single filed are in figure 19, 20 and 21.

References

[1] G. Smith and R. Walker, "Automated verification and validation of signaling systems in ptc and cbtc environments."

[2] A. E. Haxthausen, J. Peleska, and R. Pinger, "Applied bounded model checking for interlocking system designs," in *Software Engineering and Formal Methods*, S. Counsell and M. Núñez, Eds. Cham: Springer International Publishing, 2014, pp. 205–220.

[3] A. Nash and D. Huerlimann, "Railroad simulation using opentrack," in *Computers in Railways IX*, 2004, pp. 45–54.

[4] B. S. M. Kettner and C. Eickmann, "Integrating microscopic and macroscopic models for railway network evaluation," in *Association for European Transport*, 2003.

[5] H. Koustopoulos and Z. Wang, "Simulation of urban rail operations: model and calibration methodology," in *In Transport Simulation, Beyond Traditional Approaches*, 2009, pp. 153–169.

[6] B. Adeline, P. Dersin, É. Fabre, L. Hélouët, and K. Kecir, "An efficient evaluation scheme for kpis in regulated urban train systems," in *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*, A. Fantechi, T. Lecomte, and A. Romanovsky, Eds. Cham: Springer International Publishing, 2017, pp. 195–211.

[7] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker, "Automated verification techniques for probabilistic systems," in *Formal Methods for Eternal Networked Software Systems (SFM'11)*, ser. LNCS, M. Bernardo and V. Issarny, Eds., vol. 6659. Springer, 2011, pp. 53–113.

[8] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, Sep 1994. [Online]. Available: https://doi.org/10.1007/BF01211866

[9] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. Program. Lang. Syst.*, vol. 8, no. 2, pp. 244–263, Apr. 1986. [Online]. Available: http://doi.acm.org/10.1145/5397.5399

[10] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[11] ——, "Game-based abstraction for Markov decision processes," in *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*. IEEE CS Press, 2006, pp. 157–166.

[12] C. Dehnert, D. Gebler, M. Volpato, and D. N. Jansen, "On abstraction of probabilistic systems," in *ROCKS*, 2012.

[13] "Glasgow subway webpage," http://www.spt.co.uk/subway/, accessed: 2018-04-22.

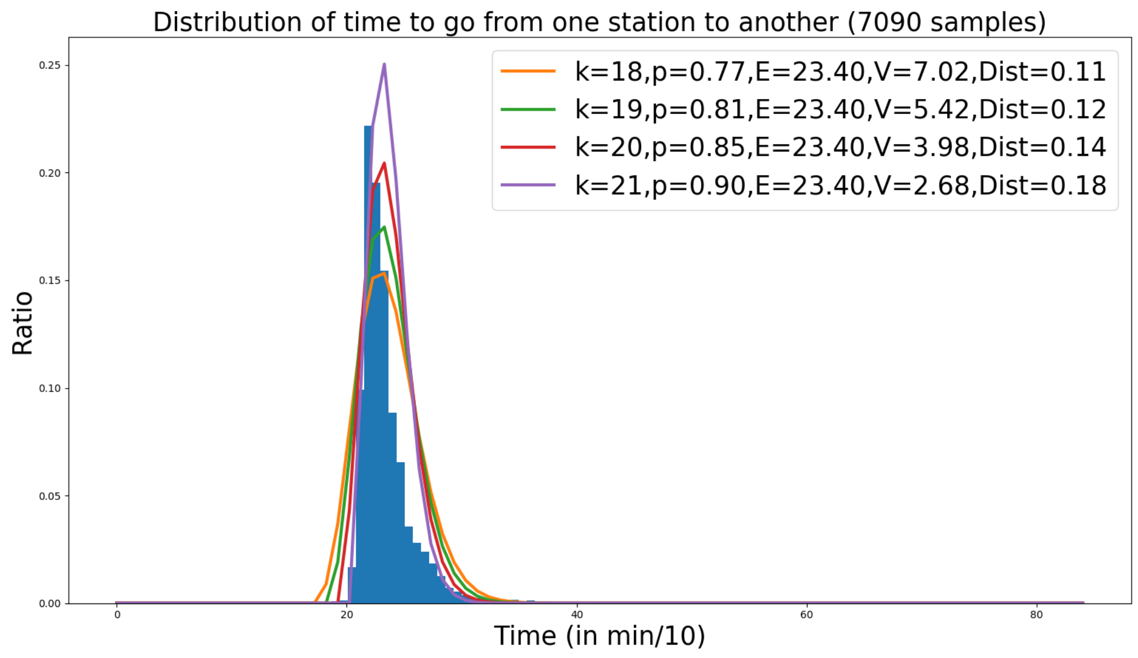

Figure 18. A distribution of the time needed to go from one station the next one. *Dist* measures the distance between the curves and the blue histogram: $Dist = \sum(curve(i) - blue\_dist(i))^2$

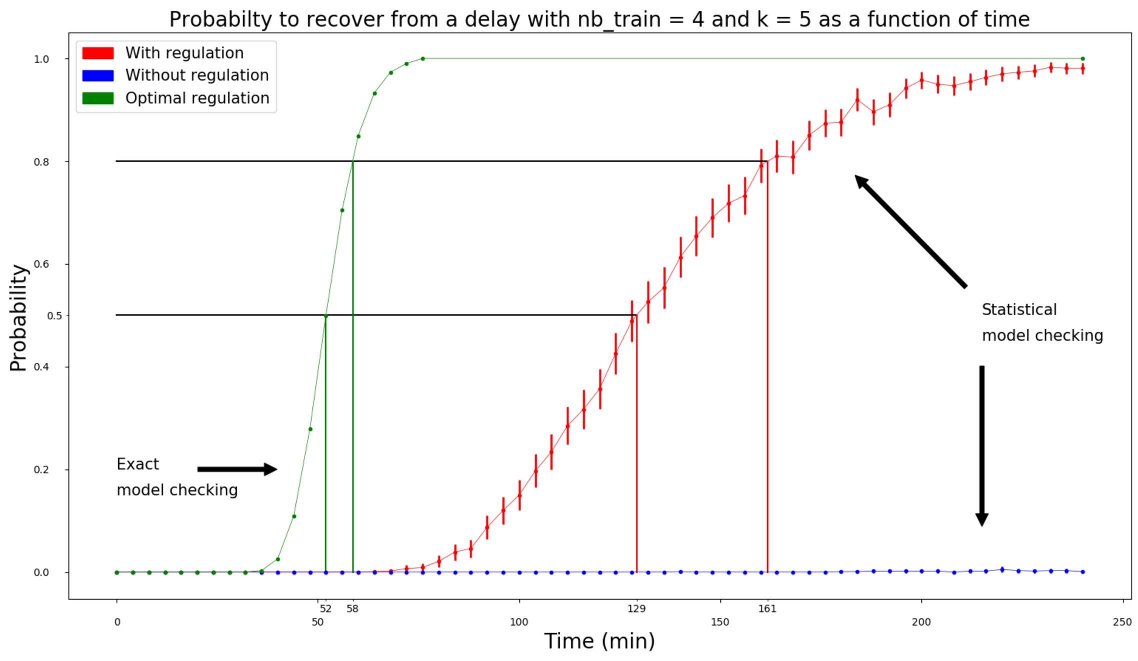

Figure 19.

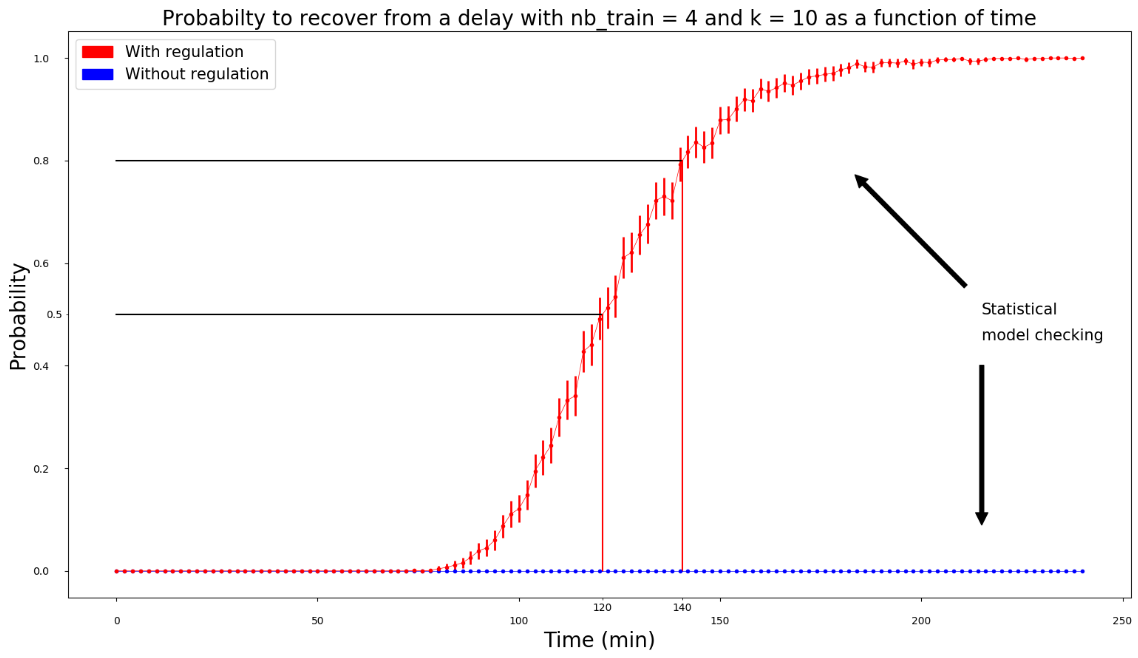

Figure 20.

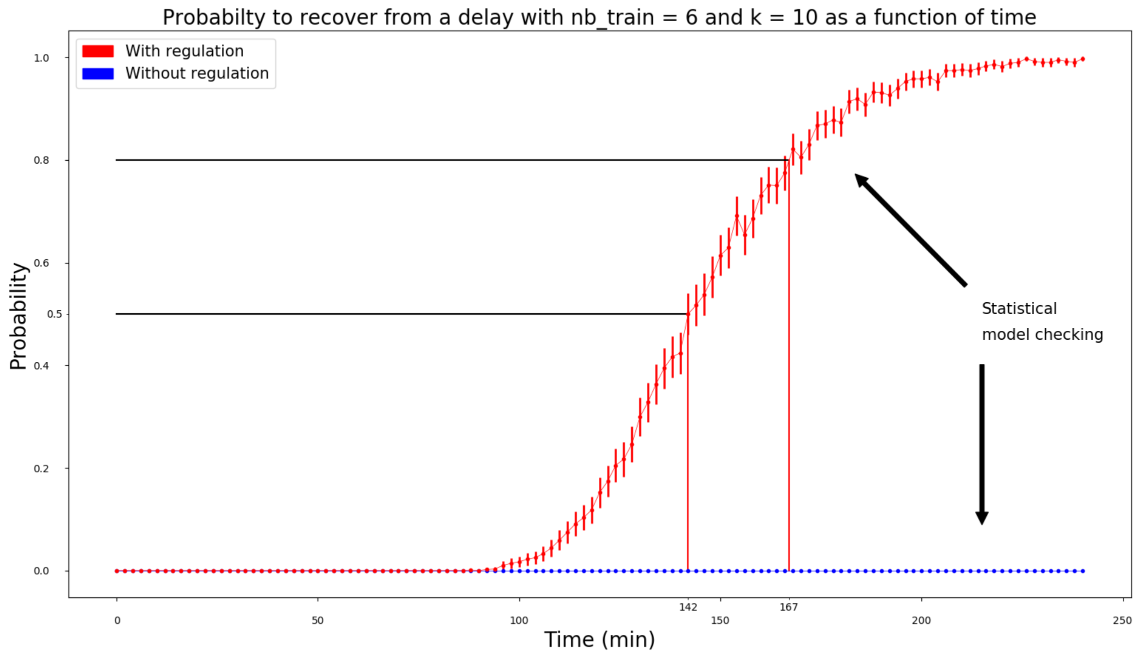

Figure 21.