Optimization of traffic management with learning machines.

Eric Fabre[1], Loïc Hélouët[1], Antoine Thébault[1]

University Rennes, Inria, CNRS, IRISA, France,
eric.fabre@inria.fr,loic.helouet@inria.fr, antoine.thebault@inria.fr

Abstract. This paper considers traffic management in metro networks, and techniques to allow fleets of trains to recover from bunching situations. We address this problem as a controller synthesis question. Controllers decide online train speeds and dwell durations to optimize the time needed to return to an ideal status of a network. We consider a formal model for metro lines based on a variant of Petri nets, and take as objective the minimization of the time needed to recover from a congestion. Though this is a standard timed game, the size of the state space for this model forbids standard synthesis techniques.
We hence address the control question with ad-hoc local controllers, that take their decisions from neighbourhood of a train and ignore the rest of the network. We first propose a local controller that balances the headways w.r.t. the preceding and successor trains at each stop. The returned solution is obtained as the minimum of a quadratic cost function. We then replace the quadratic function minimization by a decision returned by a neural network trained on our metro model, and compare performances of both types of controllers on a case study.

1 Introduction

Many real-life systems are not entirely autonomous. They need corrective mechanisms to work properly, stay in a set of expected behaviours and meet performance objectives. This is the case of metro networks, where aleas such as weather change, material failure, door incidents... can delay a train at any instant. The role of a controller is to choose the most appropriate input for the system under control to meet an objective. Some safety problems can be solved by synthesizing a controller that avoids bad configurations, some performance objectives can be enforced by synthesizing a controller that achieves the best possible reward. Though controller synthesis applies for models of small size, its cost is prohibitive. For a simple model such as priced timed games with one clock, the cost of ϵ-optimal controller synthesis is exponential in the number of states [4,7]. For metros, models sizes rapidly exceed millions of states, as an exponential blow-up w.r.t. the number of trains or stations is unavoidable [3]. Note also that keeping a decision per state is impractical as one cannot always embed large controllers in systems. Hence, computing a state based strategy with value iteration [8,1], or algorithms for priced timed games [7] does not scale.

Addressing control of large systems such as metro networks hence calls for ad-hoc solutions, even if they are not always optimal. To scale up, one should

consider controllers that take their decision only from an abstraction of the state space of the whole system. For metros, a sensible abstraction is to consider *local controllers*, that compute the most appropriate decision from characteristics of a train and of its close neighbourhood (positions and speed). This makes sense, as trains speeds are mainly constrained by the distance to the vehicle ahead.

In this paper, we address control in a priced timed game representing the behaviour of an initially bunched metro network. We consider as quantitative objective the time needed to return to a situation where all trains maintain almost equal headways w.r.t. their predecessor. Equalizing headways is a standard way to guarantee regularity of service in stations, and this type of objective is often chosen to guide traffic at peak hours. To achieve this objective, we consider local controllers that return dwell and speed advices from speed and positions of the predecessors and successors of each train. We consider two types of local traffic management schemes. We first consider *quadratic controllers* with a strategy that minimizes a quadratic function measuring differences between headways w.r.t the preceding and successor train, and the distance to a target speed. We then consider *neural controllers* composed of trained neural networks that takes as input the parameters of a single train, of its predecessor and of its successor.

We make an experimental study of the time needed to recover from bunching with the two controllers with the MOCHY tool[1], and compare the results. The experimental setting is a model for a metro network with 30 stations, and various numbers of trains. Surprisingly, while neural net controllers take the same inputs as quadratic controllers, their convergence time is much lower.

2 A Petri net variant to model Metro lines

Let us gives an overview of the model used to simulate metro networks. In short, this model assembles a *trajectory net*, a variant of Time Petri nets close to Batch Petri nets [6], and a scheduler implementing a traffic management strategy such as adherence to a timetable. We refer readers to [11] for a complete formalization.

Definition 1. *A* trajectory net *is a tuple* $TN = (P, T, {}^\bullet(), ()^\bullet, \alpha_0, \beta_0, \Omega_0, TM, M_0)$.

- P *is a set of places that contains a subset* P_C *of control places. A* marking M *is a map* $M : P \to \mathbb{N}$ *that associate an integral number of tokens to places.* M_0 *is the initial marking of the net.*
- T *is a finite set of transitions representing arrivals and departures,*
- ${}^\bullet() \subseteq P \times T$ *is the backward incidence relation,* $()^\bullet \subseteq T \times P$ *is the forward incidence relation,*
- $\alpha_0 : T \to \mathbb{Q}^+$ *and* $\beta_0 : T \to \mathbb{Q}^+ \cup \infty$ *assign respectively an earliest and latest firing time to each* $t \in T$*. We have* $(\alpha_0(t) \leq \beta_0(t))$*.* $\Omega_0 : T \to CDF$ *associates to every* $t \in T$ *a cumulative distribution function* $F_t : [\alpha_0, \beta_0] \to [0, 1]$
- TM *is a traffic management algorithm that can add a token in an empty control place or change values for* α_t, β_t, F_t*'s at any instant.*

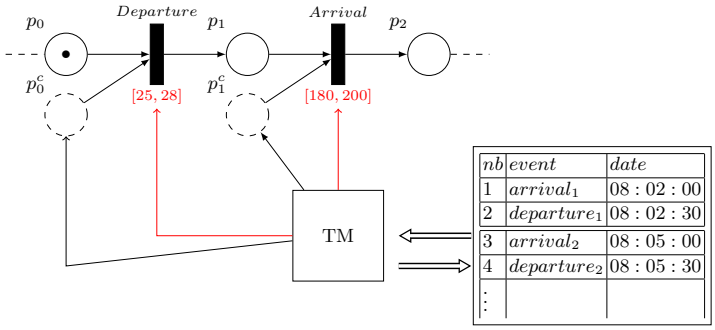

Fig. 1. A simulation model for a metro network with traffic management.

We restrict to cyclic networks (this setting is common to many metro lines). With this restriction, every place has a single transition in its postset, and every transition has a single place of $P \setminus P_C$ in its postset. For each place representing a track portion, we denote by $l(p)$ the length of the track represented by p. In a nutshell, the behavior of a trajectory net is the following. A transition represents an action of a transport network, for instance an arrival or a departure of a train. A place $p \in P \setminus P_C$ represents a track portion or a quay in a station, and a place $p^c \in P_C$ a condition needed to allow a transition (departure or arrival). A transition can fire if its preset is completely filled, and if the consumed token in $p = {}^\bullet(t) \cap (P \setminus P_C)$ has spent a sufficient amount of time in t. This duration is the value sampled from an interval $[\alpha(t), \beta(t)]$ according to a probability distribution with cumulative distribution function F_t when the token consumed has entered place p. The interval $[\alpha_0(t), \beta_0(t)]$ is the initial range of values attached to transition t, and $\Omega_0(t)$ the distribution on this interval, but traffic management can update $\alpha(t), \beta(t)$ and F_t to model speed/dwell advices, as well as aleas (crowd, weather,...).

At a given instant, the state of a trajectory net is described by a configuration $C = (M, v)$, where M is a marking, and $v : P \times \mathbb{N} \to \mathbb{R} \times \mathbb{R}$ assigns a time to arrrival/departure and a position to each token in a place. Each token in a plain place represents a train, and each token in a control place represents an order (departure order or authorization to enter a station) given by the traffic management. Let $v(p, i) = (d_i, x_i)$, and let p represent a track portion. Then, the i^{th} token in place p represents a moving trains, at distance x_i to its arrival, and which trip should last d time units. The current speed s_i of a train i is $s_i = x_i/d_i$. Similarly, if $v(p, 1) = (d, 0)$, where p represents a quay, we are in a situation where a train is in station, and will be ready to leave in d time units. From a configuration, one can easily compute the distance $D_{i,j}$ between trains i and train j by summing up distances to arrival and track lengths. A transition can fire:

– if its preset is filled. This mean in particular that a train is ready to arrive in station or to leave it. This also means that the arrival/departure occurs at

[1] https://adt-mochy.gitlabpages.inria.fr/mochy/

the date planned by Traffic Management or later, i.e. that all control places in the preset of a transition have been filled.
- if the time to arrival or time to departure of one train is 0,
- if some security constraints are met w.r.t its postset. We do not enter into details, but in short, these security constraints impose that only one train stays at a quay in a station, and that a sufficient safety distance D_{safe} is preserved between trains to allow emergency braking. Hence, even if a train is ready to leave at the scheduled time, it could be delayed for safety reasons.

Place and transitions represents the physical part of a transport network, and TM represents the traffic management policy: It can be any algorithm, with its own memory, clocks, a representation of a timetable... that schedules departures and gives dwell time and speed advices. More formally TM can update the intervals $[\alpha(t), \beta(t)]$ and distributions F_t of transitions, which amounts to setting target speeds and dwell durations. It can also put tokens in control places at chosen dates, to delay departures and arrivals. An example of policy is to adhere to a timetable. In this case, trip and dwell durations attached to transitions are computed to schedule the next arrivals or departures as close as possible to the dates specified in a timetable, and the interval $[\alpha(t), \beta(t)]$ reflects the possible perturbations around the chosen date. Tokens are put in control places at planned departure/arrival dates. Building an appropriate traffic management algorithm is hence an optimization question in a timed and probabilistic model.

Firing a transition simply follows the usual firing rules of Petri nets, and results in a new sampling of trip/dwell duration for tokens entering a new place. Firing of a transition t_i in a configuration $C = (M, v)$ results in a token move. Tokens from the preset of t_i are consumed, and a new token is created in the place $p_j = (t_i)^\bullet$. We write $M[t_i > M'$ if M' is the new marking obtained after firing t_i. Let $t_j = (p_j)^\bullet$. The time to arrival/departure for the token moved from ${}^\bullet(t_i) \cap (P \setminus P_C)$ to $(t_i)^\bullet$ is sampled from an interval $[\alpha(t_j), \beta(t_j)]$ and then decreases as time elapses. When p_j represents a station, t_j is a departure, the sampled time represents an advised dwell time for a train in configuration C plus a random delay due to incidents (for instance doors problems due to passengers misbehaviours). When p_j represents a track portion, the sampled time represents an advised duration for a trip to the next station, with random perturbations (e.g. due to bad weather conditions) along a that trip.

We refer to [11] for more details on trajectory nets. For the rest of the paper, it will be sufficient to consider them as a generators for timed sequences of events representing operations of a metro network for a given period. Their underlying semantics is a Markov Decision Process with train positions and speeds as states, dwell and speed advices as actions, and time to the next event as reward.

Definition 2. *A continuous Markov Decision Process is a tuple* $\mathcal{M} = (L \times X, A, W, \Delta)$ *where* L *is a finite set of locations,* $X \subseteq \mathbb{R}^k$ *is a* k*-dimensional set of states,* A *is a finite set of actions,* $W : L \times X \to \mathbb{R}$ *associates a reward to every state,* $\Delta : L \times X \times A \to Dist(X)$ *is a transition relation, that associates to a location* $l \in L$*, state* $x \in X$ *and action* $a \in A$ *a distribution defining the*

probability $\mathbb{P}((l', x') \mid (l, x, a))$ *to move to* $(l', x') \in X$ *from* (l, x) *when action* a *is chosen.*

A *run* of an MDP is a sequence of states $\rho = (l_1, x_1)(l_2, x_2)\dots(l_k, x_k)$. The reward of a run is the value $W(\rho) = \sum W(l_i, x_i)$. A run is *successful* w.r.t objective $G \subseteq X$ iff $x_i \in G$ for some index i. A *strategy* in an MDP is a function $\pi : (L \times X)^* \to A$ that associates a particular action to a sequence of states. We denote by Str the set of all strategies. A strategy is memoryless iff the choice of the next action played after a prefix $\rho_{[0..i]}$ depends only on the last position (l_i, x_i). A run *conforms* to strategy π iff at every step i, $\mathbb{P}((l_{i+1}, x_{i+1}) \mid (l_i, x_i), \pi(\rho_{[0..i]})) > 0$. We denote by $\mathcal{R}_{\pi,\leq K}(\mathcal{M})$ the set of runs of $\mathcal{M}$ with reward smaller than K that conform to π.

Finding an traffic management that minimizes the accumulated reward for a given objective amounts to finding $\arg\min_{\pi \in Str} \left(\mathbb{E}_{\rho \in \mathcal{R}_{\pi,\leq K}(\mathcal{M})}[W(\rho))] \right)$. In a discrete setting, this problem can be solved with positional strategies, using value iteration. In a continuous setting, one can approach the optimal value either through a first step of discretization of distributions followed by a standard value iteration process, or through lazy approximation techniques [13]. Yet, these approaches remain costly, and rapidly limit the size of systems that can be considered. For metro networks, the number of possible markings is already exponential in the number of trains K_{tr}, or in the number of stations, calling for ad-hoc solutions.

3 Quantitative objectives

A standard requirement in metro regulation is to achieve quantitative objectives, measured during long periods of network operation. These objectives can be punctuality, mean duration of trips from one end of a network to the other end, regularity of service during peak hours,.... Many quality criteria are standardized by the International Union of transports [17]. They are also central in the design of soft contracts between cities stakeholders and operators. A way to meet quality objectives is to build a timetable, and try to meet all its deadlines. However, due to random delays, this policy is not always the optimal solution to optimize a performance indicator. We take as objective the regularity of train service i.e. ensure a train departure from each station every x minutes. If we consider a metro networks as a closed loop of size Ns kms, the ideal way to achieve this regularity is to have an equal spacing $d_m = \frac{Ns}{K_{tr}}$ meters between trains, and let them run at a nominal speed $S^0 = \frac{d_m}{x}$ km/h. Further, all trains keep a safety distance D_{safe} w.r.t their predecessor.

The worst situation for a regularity objective is the so-called "bunching" situation [2], where all vehicles are located close to each other in a part of the network. When such a situation occurs, stations alternate phases where they are visited at a frequency higher than planned, followed by long periods where no train arrives. As a consequence, service is not regular, and the number of passengers in stations grows. Passengers then rush in the first train that alights (it is the train ahead of the fleet), possibly causing door incidents, delaying trips, and hence worsening the situation.

Let $C = (M, v)$ be a configuration with K_{tr} trains. We let s_i denote the speed of train i, and $D_{i,i+1}$ denote the distance between train i and its successor, namely train $i+1 \mod K_{tr}$. Similarly, we let $D_{i,i-1}$ denote the distance between train i and its predecessor $i-1$. At peak hours, the ideal situation is when trains locations in the network are well balanced, that is when headways between trains are identical up to small variations arising when trains stop. Then trains can run at speeds close to the targetted commercial speed S^0 and maintain regularity of service by slightly increasing or decreasing their current speed. Once a speed is chosen, we denote by H_i is the time needed for train i to move from its current position to the current position of train $i+1$, and by $H_{i-1} = \frac{D_{i-1,i}}{s_{i-1}}$ is the time needed by successor train to reach the position currently occupied by train i. One cannot maintain an identical distance between all trains because vehicles have to stop in station. During dwell time, distances among trains changes. Further, even if the optimal speed for train i is fixed, train movements are subject to random perturbations, and hence speeds have to be constantly adapted. Ideally, the difference between $D_{i,i-1}$ and the ideal distance d_m should not exceed the distance ran at commercial speed during a normal dwell time. The interesting information in a configuration is not the average distance $\bar{D}$, that is always equal to d_m, but rather the standard deviation. In the experiments shown later, we will consider a traffic management for a bunched sample circular network with 30 stations and up to 50 trains. This is a network of realistic size : its dimensions are close to existing networks such as the outer loop of Santiago's Metro L1 (19.3 km and 27 stations). We fix a distance of 600 meters between stations (the length of a full loop is hence 18km), and a target commercial speed of 500 m per minute ($30km/h$).

We will say that a configuration C is *balanced* iff the standard deviation is smaller than a fixed threshold. We take as threshold of 300 meters, which is half of the distance between two stations, and less than the distance ran by a train during at normal speed during a standard dwell duration. Hence C is balanced iff $\sqrt{\frac{\sum_{i=1..K_{tr}} (D_{i,i-1} - \bar{D})^2}{n}} \leq 300$, and we take as objective the set G^{eq} of all balanced configurations.

Our objective is to minimize the time needed to recover from a bunching situation with the help of a traffic management algorithm. There are many way to get back to an balanced configuration, but some strategies are not acceptable for users. Consider for instance a situation where K_{tr} trains are in a bunched configuration, i.e. in a situation where trains are positioned at decreasing distances $x_1, \ldots x_{K_{tr}}$, but are only separated by their safety distance D_{safe}, i.e. $x_1 - x_{K_{tr}} \sim K_{tr}.D_{safe}$. We target configurations where every $x_i - x_{i+1}$ is close to d_m. One possibility to get to an equilibrium is to stop all trains, then send train 1 at distance $x'_1 = x_k + Ns - d_m \mod Ns$ and stop it, then send train 2 at distance $x'_2 = x_k + Ns - 2 \cdot d_m \mod Ns$ and stop it, etc... Obviously, this is a bad solution, because trains (with their passengers) are stopped for a long time, and do not provide any transport service. Further, crowds continue to grow in stations until equilibrium, which is likely to cause new delays.

Stopping trains for a long time is not an acceptable solution, so we have to perform bunching recovery while trains are moving. To equalize headways eventually, the traffic management algorithm will compute, as soon as a train i enters a station k a duration $Ttb(i,k)$ corresponding to the time prescribed to move to the next station. Obviously, $Ttb(i,k)$ cannot take any value: one has to consider a minimal dwell time in stations, and also the minimal (resp. maximal) speed of trains. Once $Ttb(i,k)$, it is used to assign a dwell time and a trip duration (or equivalently assign a speed) to train i. These advices are then realized up to perturbations specified in the model.

In the rest of the paper, we propose two ways to compute $Ttb(i,k)$. The first one is to see this value as the optimal parameter value to minimize a quadratic function that increases with the standard deviation w.r.t mean distance, and with the distance of the fixed speed to target speed. The second approach uses neural networks to learn a function that returns an optimized value for $Ttb(i,k)$.

4 Sum of squares minimization

Let us first consider a traffic management algorithm that returns as value for $Ttb(i,k)$ the duration a trip of train i from station k to $k+1$ that minimizes the following function:

$$J(U_{i,k}) = ||H_i - H_{i-1}||^2 + \alpha \cdot ||U_{i,k} - U_k^0||^2 \tag{1}$$

$U_{i,k}$ is the parameter we are looking for, namely the time needed to complete the trip from station k to station $k+1$.This is equivalent to chosing a speed for train i along the next track. H_i is the time needed for train i to move from its current position to the current position of train $i+1$ when $U_{i,k}$ is the time needed to move from station k to station $k+1$, $H_{i-1} = \frac{D_{i-1,i}}{s_{i-1}}$ is the time needed by successor train to reach the position currently occupied by train i when running at speed s_{i-1}. U_k^0 is the nominal time for a trip at nominal commercial speed on that track. If the distance from station k to station $k+1$ is L_k, then $U_k^0 = \frac{L_k}{S^0}$.

The value $J(U_{i,k})$ is hence a weighted sum of the squared difference between H_i and H_{i-1} and of the squared difference between $U_{i,k}$ and U_k^0. The first part of the definition $||H_n - H_{n-1}||^2$ is chosen to minimize the headways differences for the current train considered, and the second part $||U_{i,k} - U_k^0||^2$ to minimize the distance to a target commercial speed. The relative importance of both parts is given by weight α. Our objective is to find the travel time $U_{i,k}$ from k to $k+1$ (or equivalently the speed s_i) that minimizes the value for $J(U_{i,k})$. The minimal value for $J(U_{i,k})$ and the value of $U_{i,k}$ that minimizes it are unique and can easily be computed.

Proposition 1. *Let C be a configuration of $\mathcal{N}$, i be a train number, k be a place index. Then*

$$Ttb(i,k) = \frac{\frac{D_{i+1,n}.D_{i,i-1}}{L_k.S_{i-1}} + \alpha U_k^0}{(\frac{D_{i+1,i}}{L_k})^2 + \alpha}$$

Notice that to compute the optimal $U_{i,k}$, the speed s_{i-1} needs to be known. When the preceding train $i-1$ is stopped in station, one can only estimate its

future speed. The best option is estimate that train $i-1$ will leave for its trip to the next station at nominal speed S^0 and compute $U_{i,k}$ with that speed.

The traffic management algorithm returns a advice $Ttb(i,k) = \arg\min J(U_{i,k})$ which is the ideal time to get to the next stop in order to reduce the differences between the headway H_{n-1} w.r.t. the train ahead of train n and the headway H_n w.r.t the train travelling immediately after train n, while staying as close as possible to the target trip duration U_k^0. From a more operational point of view, the value of $Ttb(i,k)$ is converted into a pair of advices (d_{dw}, v) where d_{dw} is a dwell time and v is a speed advice. Let $d_{tr} = \frac{v}{D(k+1)}$. Values d_{dw} and v are chosen to ensure that the time elapsed between arrival of a train at station k and arrival at the next station, i.e. $d_{dw} + d_{tr}$ is equal to $Ttb(n,k)$. As perturbations occur the actual dwell and trip durations realized are $d_{dw} + \psi_{dw}$ and $d_{tr} + \psi_{tr}$ where $\psi_{dw} \in [\alpha_{dw}, \beta_{dw}]$ and $\psi_{tr} \in [\alpha_{tr}, \beta_{tr}]$ are small positive random perturbations. In the setting of Figure 1, this amounts to instantiating TM with a traffic management algorithm that computes $Ttb(n,k)$ and sets constraints on transitions to $[d_{dw} + \alpha_{dw}, d_{dw} + \beta_{dw}]$ and $\psi_{tr} \in [d_{tr} + \alpha_{tr}, d_{tr} + \beta_{tr}]$.

Choosing the most appropriate α: At first sight, setting $\alpha = 0$, i.e. restricting equation 1 to its leftmost term may seems a sensible choice. However, early experiments showed that without the right part of equation 1, that forces trains speed to stay close to S^0, regulation may equalize distances, but letting trains move at very low speeds. In this setting, some simulation did not even converge after a very long time. We hence slightly modified the experimental setting, forcing trains to move at at least 260m/min when safety allows it.

Figure 2 shows the convergence time (in minutes) for a number of trains ranging from 5 to 50 and several values for parameter α ranging from 0 to 10. These results have been obtained through simulations of a metro model as described in Section 2, with a traffic management algorithm choosing speeds using the quadratic optimization criterion of equation 1. The curves show average convergence time for different values of α and K_{tr}. For each pair α and K_{tr}, we sampled 100 runs of our metro network, starting from a initially bunched situation, and ending in balanced configuration, with a limit of 20000 steps to equalize. All curves have a bell shape, and close equalization times for very small or very large fleets. For small fleets, equalizing distances is rather easy: as there only a few trains, metros can use the full range of speed values to move away from their successor, and hence distances equalize rather fast. When fleet size reaches the networks's maximal capacity, then trains occupy almost the whole network, and the starting bunched initial position of trains is almost balanced. Traffic management has almost nothing to do, as safety headways requirement already forces trains to preserve the distances.

The objective of our traffic management technique is first to equalize spacing of trains, but we also need to ensure smooth operation of networks. We want to avoid "on/off"' effect, i.e., to alternate very high and very low speeds. Further, once distances are equalized, one expects the traffic management algorithm to keep the speeds of trains close to the target speed. Figure 3-left shows the average dispersion of trains speed before equalization and after equalization. Un-

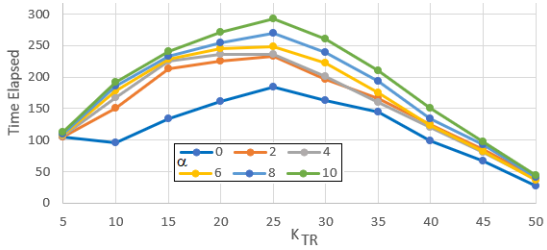

Fig. 2. Average time to reach equilibrium for several fleet sizes and values of α.

surprisingly, increasing the value of α reduces speeds dispersion. One can notice that speed dispersion is rather high before equalization, meaning that despite a "smart" traffic management, one does not fully avoid alternation of reduced speeds and accelerations. Last, Figure 4 shows the average speed of trains reached immediately after equalization. All speeds remain 3 to 5 km/h above the target speed for average fleet sizes, and the difference decreases when the size of fleet grows. Unsurprisingly, increasing the value of α can reduce the speed excess of trains by 2km/h.

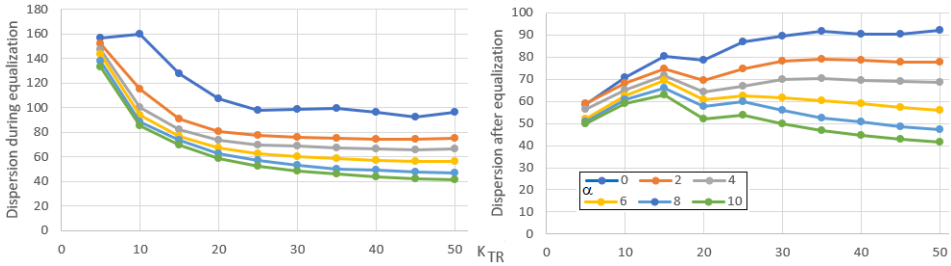

Fig. 3. Left : Dispersion of trains speed (w.r.t target speed) before equalization (left) and after equalization (right) for different values of α and fleet sizes

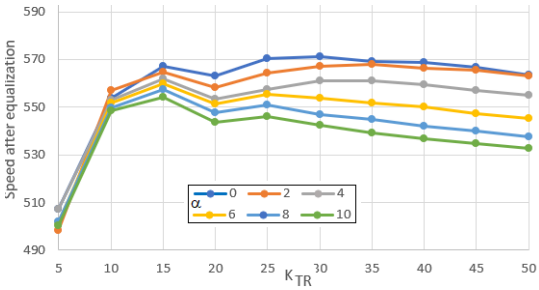

Fig. 4. Final average speed after equilibrium for different fleet sizes and α.

While traffic management with a quadratic function allows recovery from bunched situations, this strategy is not optimal. Indeed, for a given train i, this traffic management will return the same advice for all configurations where trains $i-1$ and $i+1$ are at similar distance with similar speed, regardless of positions ans speeds of other trains. The strategy implemented with the help of a quadratic function minimization is hence a strategy on an abstraction of the space of all configurations of the network. It is hence not optimal.

5 Minimization with Neural Networks

As shown in section 4, local control with a traffic management that uses minimization of quadratic function $J(U_{i,k})$ is not optimal. The question is hence whether there exists other (local) TM algorithms that perform better than this solution, i.e. reach a balanced configuration faster than the minimization solution. To answer this question, we use the advices returned by a trained neural network to help the traffic management return a dwell and speed advice. This setting is used in this section to evaluate pertinence of several parameters set before training our neural nets. We use a simple model, namely feed-forward neural networks. This type of network is known to be powerful enough to simulate any non-linear function [14]. This is however a purely theoretical result, as the number of neurons needed to simulate a complex polynomial grows fast with the complexity of the function and with the required accuracy. Yet, this is a good indication that traffic management with neural networks should at least be able to reproduce the results obtained with a quadratic function.

Definition 3. *A (feed-forward)* Neural Network *with L layers is a tuple $NN = (N, B, I, O, \ell, C, \Phi)$ where N is a set of neurons. In particular, N contains a set of bias neurons B, a set of input neurons I, and a set of output neurons O. Bias neurons have no input connection, and always produce output 1. $\ell : N \to L$ associates a layer to each neuron. Each layers contains a bias neuron. $C \subseteq N \times \mathbb{R} \times N$ is a set of connections among neurons of successive layers. Φ associates an activation function ϕ_n to every neuron $n \in N$.*

The basic computation unit of a neural network is the artificial neuron. An *input neuron* $n_i \in I$ reads a particular input x_i and returns the output $y_i = \Phi(n_i)(x_i)$. A *bias* neuron n_i returns a constant output $y_i = 1$. For all other neurons, the value returned by $n_i \in N \backslash (B \cup I)$ depends on the outputs produced by its predecessors. Let $\{(n_1, w_{1,i}, n_i), \dots (n_k, w_{k,i}, n_i)\}$ be the set of connections to n_i. The *potential* of neuron n_i is the value $a_i = \sum_{j \in 1..k} w_{j,i}.y_j$, and the output of n_i is the application of its activation function to a_i, i.e. $y_i = \Phi(n_i)(a_i)$. Activation functions can be linear functions, threshold, RELU functions,...For all neural networks of this paper, we will use the tanh activation function. Figure 5 gives a graphical representation of a neuron.

The functions computed by a neural network are the maps relating a set of inputs $x_1, \dots, x_I$ to the set of outputs $y_1, \dots y_O$ computed by output neurons after propagating values of neurons outputs layer after layer from input neurons to output neurons of the net. Figure 6 represents a complete feed-forward neural network with three layers, 6 inputs neurons and one output neuron. We denote by $n_{i,j}$ is the neuron j of layer i, and by b_i is the bias neuron of layer i, with a constant output equal to 1.

We trained neural net with 3 hidden layers of 10 neurons to learn function returning $Ttb(i)$ and convert it into a dwell and speed advice used for traffic management. Increasing the number of layers or the number of neurons per layer did not give better results. We took several inputs $x_1, \dots x_6$ to compute the value of $Ttb(i)$ obtained from the output y of output neuron $n_{4,1}$. Due to

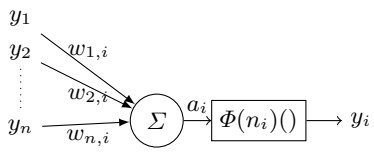

Fig. 5. A neuron n_i

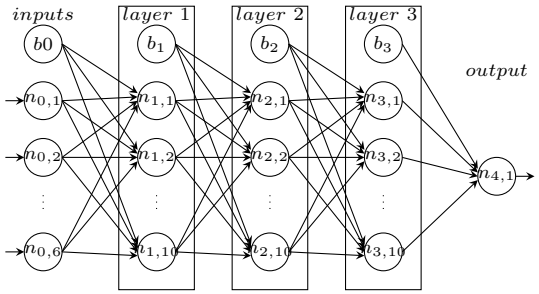

Fig. 6. A feed-forward Neural Network with 6 inputs, 3 layers of 10 neurons, and a single output neuron.

the use of the tanh, we have $y \in [-1, 1]$, while the Ttb takes value in an interval $[\alpha_k = \frac{D(p_k)}{v_{max}}, \beta_k = \frac{D(p_k)}{v_{min}}]$. We hence set $Ttb = \alpha_k + \frac{\beta_k - \alpha_k}{2} \cdot (1 + y)$, and then compute a dwell time and a speed as in Section 4. The overall simulation setting is represented in Figure 11 in Appendix C.

Training a neural net: As already mentioned, feed-forward networks are powerful, but the complexity of the simulated functions depends on the number of neurons allowed. As one cannot explore arbitrary architectures of arbitrary sizes, one usually fixes a number of layers, their composition, inputs and outputs, and train this neural net by modifying weights of connections. We hence started with a fixed architecture of neural network, represented in Figure 6-b, with 6 inputs, 3 hidden layers of 11 neurons (one bias neuron and 10 standard ones) and one output. The six inputs chosen are $D_{n-1,n}$, $D_{n-2,n}$, $D_{n+1,n}$, $D_{n+2,n}$, V_{n-1} and V_{n+1}.

A crux in training is to evaluate the quality of a neural net. One possibility is to use the average time $te(NN)$ needed to reach a balanced configuration when NN is used as speed advisor in traffic management. This criterion does not consider the speed maintained after equalization. A second possible criterion is $Eff_\beta(NN) = te(NN) + \beta \cdot |S^0 - \bar{S}^=|$, where $\bar{S}^=$ is the average speed of trains during the 1000 steps following an equalization, and parameter β gives a relative importance to adherence to the target speed S^0. We use the first criterion to train neural nets an discuss in next section the effect of speed adherence on selection of nets and convergence times.

We randomly created 1000 initial neural nets with the architecture presented above, and distinct weights. We selected the best candidates, and then improved

them with successive mutations. Each mutation was an individual mutation of one weight by ± 0.1. Best candidates were selected among nets that successfully brought the system in a balanced configuration in less than 10000 steps, and let trains run at an average speed greater than $500m$ per minute (30km/h). In the remaining nets, the 5 best candidates were selected according to the efficiency $te(NN)$, computed by simulating runs of the metro network. The process was repeated with mutations of the 5 best neural nets, and stopped when no better mutant was found. The convergence was rather fast, as no new efficient mutant appeared after 5 iterations of the learning cycle.

This experiment was also carried out with simpler neural networks, with two inputs: the distance $D_{n-1,n}$ to the previous train, and the distance $D_{n+1,n}$ to the next train. We also tested networks with four inputs $D_{n-1,n}$, $D_{n-2,n}$, $D_{n+1,n}$ and $D_{n+2,n}$. Unsurprisingly, the performance of best networks with 6 inputs exceeded that of best networks with two and four inputs. A crucial question is the size of fleets used when training neural nets. As explained later, this size is a crucial parameter of the training environment. Indeed, for safety reason, trains have to maintain in all situations a safety distance D_{safe} w.r.t their predecessor. When many trains are used $K_{tr} \times D_{safe}$ is almost the size of the whole network, and controllers cannot choose the speed of trains. In some sense, when the network reaches it maximal fleet capacity, traffic management becomes useless because respecting headways is sufficient to guarantee equal spacing between trains. Training neural networks with large fleets produces controllers with bad performance. Now the question is whether a neural net trained with a fleet of size K_{TR} can guide efficiently a bunched network with a fleet of different size.

We have trained neural networks with varying fleet sizes ranging from 5 to 50 trains, and used then with fleets of sizes 5 to 50 to study the impact of fleet sizes on training. Surprisingly, some neural networks optimized for 5 tokens were able to control networks with different fleet sizes with good performance. Figure 7 represents the time needed to reach a balanced configuration for neural nets with 6 inputs, trained with different size fleets, and then used for fleet sizes ranging from 5 to 50. This diagram shows that neural networks have close convergence time independently from the fleet size used to train them. Surprisingly, the network trained with 5 trains has slightly better results than others. Later we will use it as one of the reference nets for performance comparison.

Figure 8 shows the time needed to equalize for a representative sample of neural networks (left) and the average speed of trains that these networks maintain after equalization. We denote by $rnx - y$ a network with x inputs trained with a fleet of size y. One can notice that in the neural network with 6 inputs trained with 5 trains seems to allow fast convergence time for any size of fleet, it does not allow to remain close enough to target speed and has lower performance than the optimal equalization with a quadratic criterion and speed adherence parameter α set to 0. This is due to the fact that distance to target speed was not used to measure efficiency of nets.

Figure 9 shows the speed dispersions for systems controlled by the same neural nets. The left part of the Figure represents the dispersion of train speeds

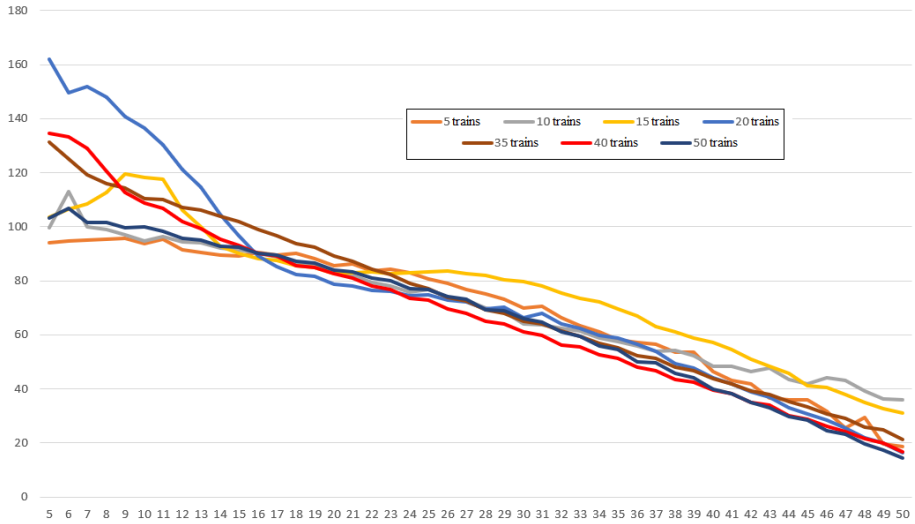

Fig. 7. Convergence time for different neural networks

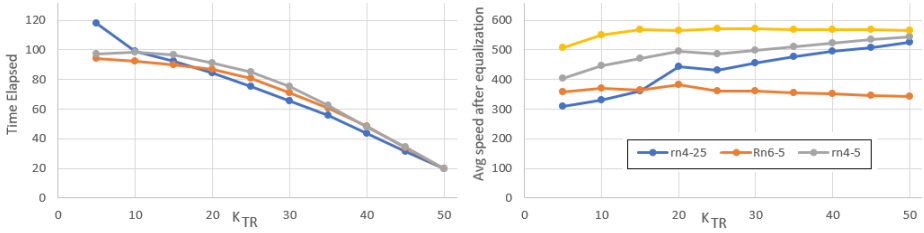

Fig. 8. Convergence time for equalization (left) and average speed of systems controlled by neural nets after equalization (right).

before reaching a balanced configuration, and the right part the average dispersion of train speeds during the 1000 simulation steps that follow equalization. One can see that neural nets trained with 4 inputs have close performance regardless of the size of fleets used to train them. Surprisingly, adding more inputs seems to impact the range of speeds at which trains are running after equalization. One hypothesis is that adding more information on distance when speed is not considered in the quality criterion used to select nets simply adds noise and minors the importance of speed in decisions.

6 Comparison of results

We can now compare the results achieved with traffic management algorithms equipped with a quadratic function minimization strategy and with neural networks. The left curves in Figure 10 show the time needed to recover from bunching with a quadratic function tuned with parameter $\alpha = 0$ (cyan), $\alpha = 10$ (green), and similar results for a neural net $RN4$-5 with 4 inputs trained with 5 trains (yellow), and $RN4$-25 with 25 trains (Orange), with a selection function

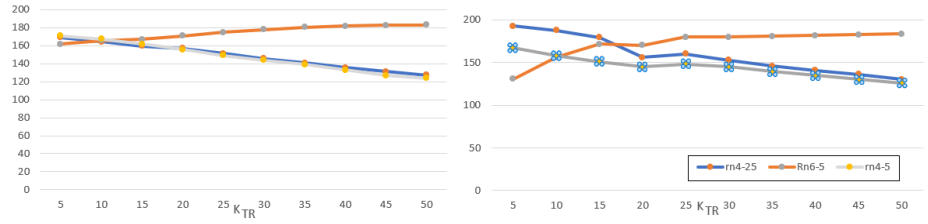

Fig. 9. Dispersion of speeds before/after equalization for the nets of Figure 8

that considers convergence time only. The last curve in dark blue represents the performance of an additional neural net $RN6$ with 6 inputs that was the best neural net selected by function $EFF_\beta()$ with $\beta = 1$, hence giving some importance to the target speed. All convergence times were obtained with a sample of 1000 runs, for each size of fleet. We have shown in Section 4 that choosing appropriately parameter α avoids situations where trains have very low speeds, but increases convergence time. Another obvious result is that neural networks outperform the results of quadratic functions in terms of convergence time. It confirms that control with a quadratic function is not an optimal strategy. An interesting information brought by the $RN6$ curve is that when distances of trains speeds to a target speed are considered, convergence time increases, and is impacted by the size of fleet. Yet, performances of $RN6$ remain better than equalization with a quadratic function.

Last, one can notice that the convergence time for all controllers is a duration smaller than 50 minutes when the size of fleet approaches the maximal capacity of the network. As already explained, this is due to the fact that bunching situations are almost balanced configurations when a network is used at its maximal capacity, as the distance between two consecutive trains is close to the safety distance. In such situations, neural networks or quadratic controllers do not have many corrections to bring, and convergence is fast.

7 Conclusion

This work has considered local control of metro networks to escape bunched situations. We have considered two versions of local controllers: the first version is a controller returning speed and dwell advices computed as a result of minimization of a quadratic function, that considers spacing among trains and distance to a target speed. The second version of controllers computes speed and dwell advices too, but from a result returned by a trained neural network. Neural networks controllers outperform quadratic ones w.r.t. the time needed to reach an equilibrate configuration. However, when selection of best neural nets uses a criterion that considers distance to a target speed, the performance of the selected nets decreases. Another lesson learned is that the number of trains used during training does not affect significantly learning nor performance of neural-based controllers. Using distance and speeds of neigbour trains as inputs can be seen as some form of abstraction. Similarly, local strategies compute a speed/dwell

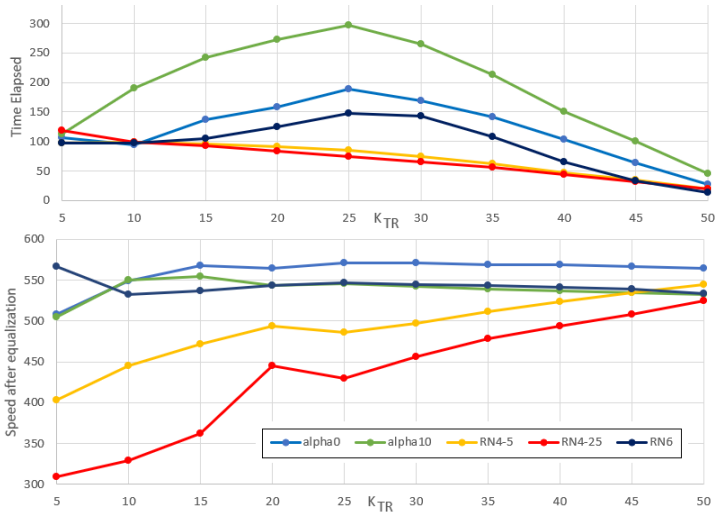

Fig. 10. Comparison of convergence times and speeds for controllers equipped with quadratic function minimization and neural nets.

advice for a single train and these strategies can only be sub-optimal. A question is whether errors made by local strategies are compensated by corrections brought by the following control actions of trains that were not considered in the neighbourhood, regardless of the number of trains in a fleet. Understanding the function implemented by a neural net, characterizing the nature of the abstraction they implement, and whether some properties of distribution, inputs and global objectives give guarantees w.r.t. optimality are issues for future work.

Related Work: Neural networks and learning techniques are nowadays widely used to solve problems such as strategy games, autonomous vehicles driving, image recognition, etc. [5] uses reinforcement learning to compute reachability strategies, and [9] proposes learning algorithms that build partitions of the state space of a continuous space Markov Decision Processes. Neural networks have also regained interest for control recently, but this model has been used for a long time. As an example, [14] trains and uses neural networks to guide trucks to a docking position. Artificial intelligence techniques have also met a recent interest in the context of urban transports. [10] proposes to use Q-Learning to choose a strategy for trips selection (with the possibility to skip stations), the ratio of passengers allowed to enter quays, with the objective to optimize passengers flows. The quality function to minimize in this Q-learning approach considers the number of passengers on quay to guide actions selection. [15] proposes a Deep Q-Learning technique where the quality function is approximated by a neural network to minimize the mean delay w.r.t. an original timetable. [12,16] also uses Q-Learning to reschedule trains in metro lines. Configurations are current positions of trains, and actions a Go/no-Go order for each train. The quality function for action selection is the probability that all trains complete their mission.

References

1. C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.
2. JJ. Bartholdi and D.D Eisenstein. A self-coordinating bus route to resist bus bunching. *Transportation Research*, 46(4):481–491, 2011.
3. N. Bertrand, B. Bordais, L. Hélouët, T. Mari, J. Parreaux, and O. Sankur. Performance evaluation of metro regulations using probabilistic model-checking. *RSS-Rail'19*, pages 59–76, 2019.
4. P. Bouyer, K.G. Larsen, Nicolas Markey, and J.I. Rasmussen. Almost optimal strategies in one clock priced timed games. In *Proc. of FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.
5. A. David, P.G. Jensen, K.G. Larsen, A. Legay, D. Lime, M. Grund Sørensen, and J. Haahr Taankvist. On time with minimal expected cost! In *Proc. of ATVA 2014*, volume 8837 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2014.
6. I. Demongodin, N. Audry, and F. Prunet. Batches Petri nets. In *Proc. of Systems, Man and Cybernetics'93*, volume 1, pages 607–617. IEEE, 1993.
7. T. Dueholm Hansen, R. Ibsen-Jensen, and P. Bro Miltersen. A faster algorithm for solving one-clock priced timed games. In *Proc. of CONCUR 2013*, volume 8052 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2013.
8. 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)*, volume 6659 of *LNCS*, pages 53–113, 2011.
9. M. Jaeger, P. Gjøl Jensen, K.G. Larsen, A. Legay, S. Sedwards, and J. Haahr Taankvist. Teaching stratego to play ball: Optimal synthesis for continuous space mdps. In *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*, pages 81–97. Springer, 2019.
10. Z. Jiang, J. Gu, W. Fan, W. Liu, and B. Zhu. Q-learning approach to coordinated optimization of passenger inflow control with train skip-stopping on a urban rail transit line. *Computers & Industrial Engineering*, 127:1131–1142, 2019.
11. K. Kecir. *Performance Evaluation of Urban Rail Traffic Management Techniques. (Évaluation de Performances pour les Techniques de Régulation du Trafic Ferroviaire Urbain)*. PhD thesis, University of Rennes 1, France, 2019.
12. H. Khadilkar. A scalable reinforcement learning algorithm for scheduling railway lines. *IEEE Trans. Intell. Transp. Syst.*, 20(2):727–736, 2019.
13. L Li and M.L. Littman. Lazy approximation for solving continuous finite-horizon mdps. In *Proc. of the* 20^{th} *National Conference on Artificial Intelligence and the* 17^{th} *Innovative Applications of Artificial Intelligence Conference*, pages 1175–1180. AAAI Press / The MIT Press, 2005.
14. D.H. Nguyen and B. Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, 10(3):18–23, 1990.
15. L. Ning, Y. Li, M. Zhou, H. Song, and H. Dong. A deep reinforcement learning approach to high-speed train timetable rescheduling under disturbances. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019.
16. D. Semrov, R. Marsetic, M. Zura, L. Todorovski, and A. Srdic. Reinforcement learning approach for train rescheduling on a single-track railway. *Transportation Research Part B*, 86, 2016.
17. UITP. Metro service performance indicators. Technical report, UITP : International Association of Public Transport, April 2011.

A MDps for Trajectory nets

Let T_{now} be the time elapsed so far, and let $C = (M, v)$ be a configuration of the net reached at that date. Form v, one can find the minimal duration d_C that must elapse before occurrence of an event. We have $d_C = \min\{v(p, i) \mid M(p) \geq i\}$. Several events can be scheduled at date $T_{now} + d_C$, so the role of TM is to choose of them, i.e. select one transition t_i to fire, update the marking of the net accordingly to obtain a new marking M', compute an advice $Adv(C)$ for the token moved from ${}^\bullet(t_i)$ to $p_j = (t_i)^\bullet$. It is a common use that metros have predetermined dwell ans speed profiles, so we assume that advices are integral durations chosen in an interval $[l_i, u_i] \in \mathbb{R}^2$. Once an advice in $[l_i, u_i]$ is chosen, if a distribution on perturbations is known, one can set interval $[\alpha(t_j), \beta(t_j)]$ and distribution F_{t_j} and sample a sojourn time for a token in place p_j. The underlying semantics of our metro model is hence a Markov decision process with continuous state space. As we are interested in objectives such as minimal time spend to reach a particular state, one can attach to every state a reward that represents the time elapsed since the last event, and hence represent a train network as a Markov Decision Process.

We can associate a MDP $\mathcal{M}_N$ to the trajectory net N that simulates the behavior of a metro network with K_{tr} trains. $\mathcal{M}_N$ records a position and a time to arrival/departure for each train. The position of a train can be recorder as a real value in $[0, Ns]$, where $Ns = \underset{p}{\in} (P \setminus P_C) \sum l(p)$. We assume an upper bound mxt on trip and dwell duration. The set of states of $\mathcal{M}_N$ is hence $[0, Ns]^{K_{tr}}.[0, mxt]^{K_{tr}}$. The set of actions is $A = T \times \mathbb{N} \cap [0, mxt]$, and for x, a fixed, $\Delta(x, a)$ is a continuous distributions, for instance of normal or Weibull distributions. The probability to move from a state (l, x) to a state (l', x') via $a = (t_i, v)$ is 0 if l' is not the marking obtained from l after firing t, and $\Delta(x, a)(x')$ if $l[t_i > l'$. Further, the new state x' affect the coordinates of the state attached to the train that moved, and leaves the other components of state x unchanged. We assume an objective given a set of states $G \subseteq X$, and use rewards to measure time elapsed from the beginning of a run to the arrival in a goal state of G. We associate to each configuration $C = (M, x)$ such that $X \notin G$ a reward $W(x) = d_C$ and to other states a reward 0.

B Proof of Proposition 1

Proposition 1: Let C be a configuration of $\mathcal{N}$, i be a train number, k be a place index. Then

$$Ttb(i, k) = \frac{\frac{D_{i+1,n}.D_{i,i-1}}{L_k.S_{i-1}} + \alpha U_k^0}{(\frac{D_{i+1,i}}{L_k})^2 + \alpha}$$

Proof. We choose as $Ttb(n, k)$ the value of $U_{i,k}$ that minimizes $J(U_{i,k})$. We develop $J(U_{i,k})$, and we obtain:

$$J(U_{i,k}) = ((\frac{D_{i,i+1}}{L_k})^2+\alpha)(U_{i,k})^2 - 2(\frac{D_{i,i+1}.D_{i,i-1}}{L_k.S_{i-1}}+\alpha U_k^0)U_{i,k}+(\frac{D_{i,i-1}}{S_{i-1}})^2+\alpha(U_k^0)^2$$

$J(U_{i,k})$ is hence a quadratic function of the form $a.x^2 + b.x + c$, with $a = (\frac{D_{i+1,i}}{L_k})^2+\alpha$, $b = -2(\frac{D_{i,i+1}.D_{i,i-1}}{L_k.S_{i-1}}+\alpha U_k^0)$ and $c = (\frac{D_{i,i-1}}{S_{i-1}})^2+\alpha(U_k^0)^2$. a is strictly positive, so the minimal value for $J(U_{i,k})$ is obtained at the lowest point of the function's parabola. The absiscae of this point is $x = \frac{-b}{2a}$. □

C The experimental setting to train Neural Network controllers

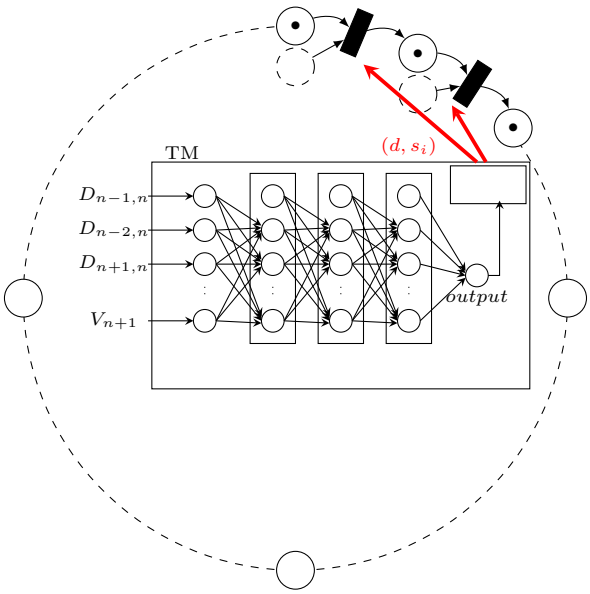

Fig. 11. Simulation setting with neural networks.