Smart Traffic Lights
Internet of Things 2025

Andrea Donato, Simone Tablò

November 22, 2025

{donato.1808606, tablo.1800830}@studenti.uniroma1.it

Abstract

In this work we present a Q-Learning based approach to solve the proposed smart traffic lights optimization problem. This includes comparisons with the classical non-smart time-based solution and adaptations to non-stationary probability distributions. In this theoretical, centralized scenario we find improvements in lowering traffic congestion up to 70% with respect to the non-smart scheme. Also, we found that Q-Learning is able to manage bad-behaving drivers, in order to lower the (huge) impact they have on traffic congestion.

Contents

1 Introduction **3**

2 Overview of the problem **3**

3 Algorithms **4**

4 Implementation **6**
4.1 Non-stationary distributions 6
4.2 Genius drivers 7

5 Evaluation **9**
5.1 Exploration 10
5.2 A Dynamic Day 14

6 Results **16**

7 Future Works **17**

A Appendix **18**

1 Introduction

As of 2025, most traffic lights follow non-optimized synchronization schemes, typically timer-based. Even if we try to optimize the timer-based scheme itself, fluctuations may still produce unexpected results. This inevitably leads to a more severe congestion of the whole network. A different, more efficient approach would consist in dynamically switching the lights relying on real-time observations: it's the definition of a Reinforcement Learning problem.
Of course, a real-world application would require the choice of IoT devices, protocols and so on. Here, we limit ourselves to a simulation of what would happen in a real world application. While being a somewhat optimistic scenario (e.g. no potential communication issues between devices, no centralized/decentralized architecture dilemma, ...), this allows us to test the response of the system w.r.t. different input parameters. Moreover, this MDP will only involve a single couple of devices, i.e. "stop-or-go" signals for a single road intersection. This makes the simulation computationally easier while still catching the idea behind the general solution. One may extend the whole thing to multiple intersections.

2 Overview of the problem

The state of system is completely defined by three quantities, namely

- the current number of cars in the first (n_1) and in the second (n_2) queue. Then, from these two values we may obtain the total current number of cars in the whole system (N), which will be the main metric to evaluate overall congestion;
- the current state of the two traffic lights, $\texttt{TL}_1, \texttt{TL}_2 \in \{\texttt{green}, \texttt{red}\} \approx \{0, 1\}$. Since $\{\texttt{TL}_1 = 0 \Rightarrow \texttt{TL}_2 = 1\}$, $\{\texttt{TL}_1 = 1 \Rightarrow \texttt{TL}_2 = 0\}$ and no other combinations are possible, we may compress these two bits into a single one[1]. Thus, from now on $\texttt{TL} = 0$ will mean $\{\texttt{TL}_1 = 0, \texttt{TL}_2 = 1\}$ and viceversa.

At the end of the day, we get

$$s = (n_1, n_2, \texttt{TL})$$

These are the states of our MDP, now we need to define actions. In order to do that, we must note that in any case at each time step:

- a number $x_i \leftarrow X_i = \texttt{Uniform}(0, n_i^{\text{in}})$ of cars join queue i. If this operation would overflow the maximum capacity $n_i^{\max}$ of the queue i, it just ends up filling the queue;
- a number $y_i = \min(n_i, n_i^{\text{out}})$ leave the queue if and only if $\texttt{TL}_i$ is $\texttt{green}$. n_i^{green} represents the maximum amount of cars that can leave the queue per time step.

[1] Of course, one may imagine a scenario in which both lights are red. Such a thing might help in situations like the one described in Section 4.2. However, in this work we'll ignore it.

We chose to summarize the combined action of these two fluxes in a single variable

$$\Delta_i = x_i - y_i$$

That being said, at each step we may either choose or not to change the value of `TL`, meaning either performing a color switch or leaving the colors unchanged. Transactions on the Markov Chain occur even if we take no actions[2]: for instance,

$$s = (n_1, n_2, \texttt{TL}) \to s' = (n_1 + \Delta_1, n_2 + \Delta_2, \texttt{TL})$$

Of course, both Δ_1 and Δ_2 will contain the x component, while only one will present the y one. Actions are the only element on which we have control: in fact, each state of this MDP may be schematized as having only two outgoing arrows, each corresponding to an action. Then, as we can see in the previous example, each action may lead to a number of different states given by Δ_i. In other words, each action-arrow splits probabilistically. For instance, if we set

$$\left\{ n_1^{\text{in}} = 5, n_1^{\text{out}} = 3; \quad n_2^{\text{in}} = 3, n_2^{\text{out}} = 2 \right\} \quad \Rightarrow \quad \Delta_1 \in [0, 5] \quad \Delta_2 \in [0, 3]$$

and we take any action we get $6 \times 4 = 24$ reachable states, each with probability $1/24$[3]. Typically, at the beginning of a RL algorithm we initialize actions with uniform probabilities. Assuming $P(s, a_i) = 1/2$, the initial Markov Chain is such that from state s we can reach 46 different $s' \neq s$ states, each with probability $1/48$, and take the loop to $s' = s$ with probability $1/24$. While looking for the optimal action these probabilities will change, but only as a consequence of modifying $P(s, a_i)$ by updating actions' Q-Value. Updates are essentially based on rewards. A possible way to set the reward system is to consider the overall number of cars N at a given time, defining

- $N <$ `low threshold` - Low traffic, positive reward (e.g. $+1$);
- $N >$ `high threshold` - High traffic, negative reward (e.g. -1);
- Any other situation - Medium traffic, neutral reward (e.g. $+0$).

3 Algorithms

As we said, we may approach this problem via multiple algorithms.

- A simple timer-based scheme represents a somewhat worst case scenario. Since no observations on the current state of the system are performed, our only hope would be to assume to know the incoming cars' distributions for each queue, then set a proportional time for each traffic light. We'll use it as a benchmark scenario.

[2] Or better, if we take the "do-not-switch" action.

[3] This includes $s' = s$, the loop being taken if $\Delta_1 = \Delta_2 = 0$.

- If we assume to know the distributions, then the optimal solution can be found solving the associated Markov Decision Process. In a real world scenario this is highly unpractical. Again, we'll take advantage of it only as a benchmark scenario.
- Not knowing the distributions means to learn them while taking potentially sub-optimal actions, leading to Q-Learning.

Q-Learning with ε-greedy strategy is a good trade-off between optimal algorithms in the stationary case (e.g. Optimistic Initial Values + greedy strategy, usually providing faster convergence) and the need to also consider the case of non-stationary probability distributions for arrivals (4.1).
To briefly summarize the idea: we uniformly initialize the Q-Values for each action of each state, namely

$$Q_{t=0}(s, a_i) = 0 \quad \forall\, s, a_i$$

At time t, taking action a_t in state s_t will reward us with r_t and lead us to state s_{t+1}, ready to take the action a_{t+1}. Since $Q(s_t, a_t)$ represents the expected reward we may obtain by taking action a_t while being in state s_t and r_t is the actual observed one, we may update our estimation as

$$Q^{\text{new}}(s_t, a_t) \leftarrow Q^{\text{old}}(s_t, a_t) + \alpha\, \text{TD}(s_t, a_t)$$

where

$$\text{TD}(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a_{t+1}) \; - \; Q(s_t, a_t)$$

- $\gamma \in [0, 1)$ is the discount factor. $\gamma = 0$ means not taking into account the "quality" of the landing state (only caring about the action itself, instead), while $\gamma \to 1$ is a more long-term-oriented approach[4];
- $\alpha \in (0, 1]$ is the so-called step size (or learning rate), a measure of how fast we look for the optimal solution. High values[5] of α may result in wide oscillations around the optimum, while low values may result in slow convergence. A good trade-off is to let α slowly decay, meaning quickly get near the minimum and then slowing down to be sure not to just jump around it.

Once we have the Q-Values, we choose action

$$a = \arg\max_{a'} Q^*(s, a')$$

with probability $1 - \varepsilon$, and perform a random choice with probability ε, where the latter follows an harmonic decrease

$$\varepsilon = \frac{c}{c + k}$$

.

[4] $\gamma = 1$ would make sense if future rewards were finite (i.e. the process has a finite number of steps), or in particular situations in which rewards decay "fast enough" (otherwise the expected reward might diverge). That's not the case. We typically set $\gamma = 0.95$.

[5] While allowed by the mathematical setting, we ignore $\alpha > 1$. It would be a clear way to intentionally "jump around" and avoid to find the minimum.

4 Implementation

This section provides a quick overview of the code, which can be found at this link.
First of all, we initialize the parameters for the scenario we want to simulate. This includes all the values seen in section 2 (incoming/outgoing cars, traffic thresholds), but also

- the ones needed for achieving consistent simulation results: a `seed` for the pseudo-random numbers generation, the number of identical `simulations` to be performed (in order to reduce the noise of the single simulation and obtain smoother graphs), the number of `steps` for each simulation.
- the Q-Learning ones: α, $\varepsilon_{\text{start}}$ and $\varepsilon_{\text{minimum}}$ define convergence speed and stability over local minima;
- the `hours` parameter is used to implement non-stationary distributions (see 4.1).

The "physical" parameters allow us to esteem a realistic parallelism with a real world scenario: `step` represents an amount of time, and $n_i^{\text{in/out}}$ the corresponding car flux. If, for instance, we choose n-values somewhere between 1 and 10, we may associate a single `step` with a duration of $\sim 5s$. Then, simulations with `steps`$= 60$ last ~ 5 minutes. We'd set `steps`$= 720$ for an hour and `steps`$= 17280$ for a day. For an easier implementation of the non-stationary distributions section (4.1), we decided to simulate a day with 18000 steps (each corresponding to $\sim 4.8s$). We stress, anyway, the fact that these are arbitrary values depending on the choice of the number of incoming/outgoing cars at each step, together with plain common sense.

Then, we run the benchmark cases. The timer-based one needs no further actions: it is sufficient to run a simulation where lights are switched every `k` steps (`k=4` by default, corresponding to a light switch every $\sim 20s$). Of course, one may set an asymmetric time duration in order to better fit the unbalance between the queues. However, since this would mean to assume some prior knowledge of the system, we chose not to.

As for the static MDP, we first run the `mdp_value_iteration` function in order to find the stationary probabilities and extract the best policy as

$$a_s = \arg\max_{a'} V(s, a')$$

Then we proceed by simulating the arrivals with the `mdp_simulate` function, choosing the best action looking at the previously found policy.
At last, Q-Learning's implementation `mdp_q_learning` is a bit like merging the last two functions: values are updated as "real" traffic flows.

4.1 Non-stationary distributions

The main problem while trying to optimize a timer-based or a MDP model is that the arrivals' distributions change over time. Q-Learning, on the other hand, is perfectly capable of adapting to new situations, adjusting its policy to fit the new scenario. Here, we

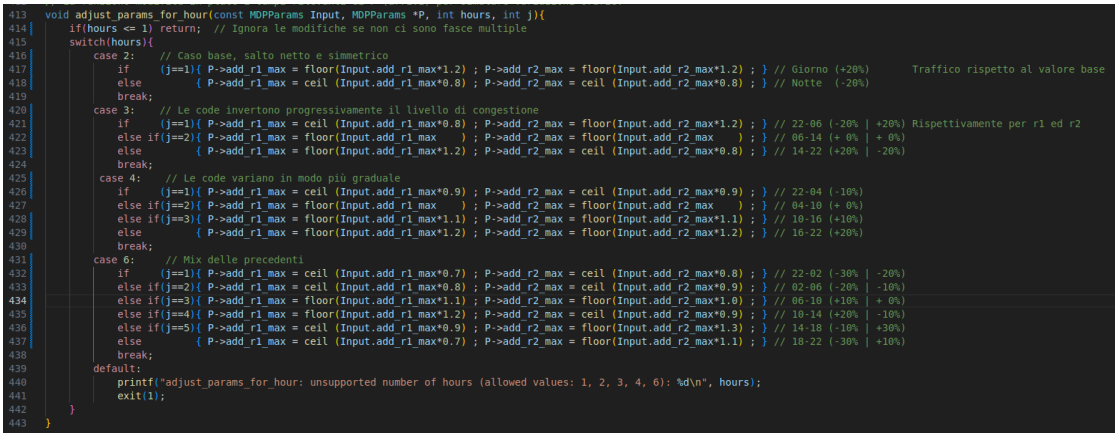

Figure 1: Traffic intensity variations depending on the time. The logic behind these particular choices for the values will be better explained in Section 5.2.

divide a single simulation (i.e. 18000 steps, a day) into multiple time zones, each with its own arrivals distribution obtained as a percentage variation w.r.t. the input values. The easiest way to visualize this scheme is to look at Figure 1.

Let's take `hours=2` as an example and look at the `adjust_params_for_hour` function. First (`j=1`), the input parameters `add_r_max` are incremented by 50%, rounded down (we'd better avoid to overload the network, where possibile). The first call to `mdp_q_learining` starts from state `s0` with uniform Q-Table values `Q0`, and outputs a state `s1` with non-uniform Q-Values `Q1` (i.e. the learned best values so far). Then the function re-adjusts the parameters, this time to half their original values rounded up (we don't want a uniformly empty input!). The second call to `mdp_q_learining` starts from state `s1` and Q-Values `Q1`, returning `s2` and `Q2`. It is straightforward to generalize.

One may notice that the last state of the day might be the input state of the new day. That's exactly how multiple simulations work in the case `hours>1`: while the stationary case is evaluated as repeated identical simulations, here we can simulate multiple days in sequence, then evaluate the generic day by taking the average over multiple days, just like in the previous cases.

4.2 Genius drivers

According to the (italian, at least) Highway Code, vehicles are permitted to enter an intersection if the traffic light is green and they are actually able to clear the intersection — that is, if they are able to avoid being blocked within it. So far, we assumed this rule to be respected. As many of us experience everyday, that's typically not the case. Such a behavior, i.e. violating this rule, has severe consequences when trying to optimize traffic flow in a potentially congested scenario. It's easy to see why:

- If rules are respected, the well-behaving driver notices that even if the light is green he has no possibility of clearing the intersection, so he stops at its traffic light. Queue `1` is blocked, but as soon as lights switch, traffic in queue `2` is free to flow.
- If rules are not respected, the genius driver stops right in the middle of the intersection. Queue `1` is blocked anyway, the only difference being that now it also blocks queue `2` as soon as the light switch occurs. Traffic flow is blocked in both directions.

Since we don't know the state of the road after the intersection[6], we modeled such a situation as follows:

1. All drivers are genius drivers (bad-behaving drivers).
2. Each car that clears the intersection increases the probability that the following one will block it (i.e., it's plausible to assume that each car may increase the population of the next queue, eventually filling it up). This probability increase is proportional to the n_i^{out} of the given queue, and it's set to take the value $\sim 63\%$ when the car flux is maximum over `4` consecutive `steps`. Whenever a switch occurs, the probability is reset to `0`.
3. At each `step`, after the probability update, a random value extraction decides whether the last car blocks the intersection or not, storing the result in a boolean value `block`.
4. If `block` is set to `1`, no cars can pass the intersection, no matter the value of `TL`.
5. At each step where `block = 1`, an "unchoking" probability increases exponentially in the number of `steps`, repeating the random value extraction and decision as described before and eventually setting `block = 0`.

This scenario will be simulated by an *ad hoc* function `geniusDrivers_q_learning`, here reported as `Genius Q-Learning` or `GQL`.

[6] It would probably depend on another traffic light, which is not included in this simulation. We'll leave it as an idea for a future work (see Section 7).

5 Evaluation

First, we run a simulation with the proposed parameters:

$$\left\{ n_1^{\max} = 40, n_2^{\max} = 25, \quad n_1^{\text{in}} = 5, n_1^{\text{out}} = 3; \quad n_2^{\text{in}} = 3, n_2^{\text{out}} = 2 \right\}$$

We also set the proposed standard thresholds $\texttt{low\_th} = 15, \texttt{mid\_th} = 30$. The outcome of this particular simulation was somewhat expected. Results are shown in Figure 2.

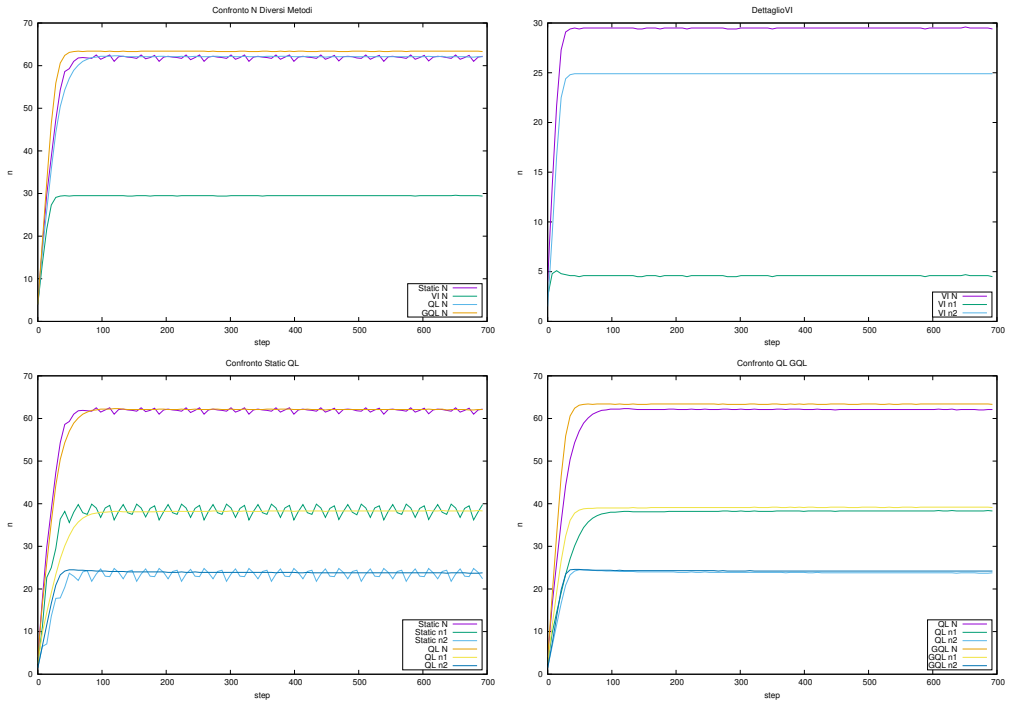

Figure 2: In a bottleneck scenario the system saturates, no matter the algorithm. The only exception is the optimized MDP, which chooses to completely block the smallest queue in order to let traffic flow in the largest one.

Let us analyze why the system saturates. While it is true that the expected values for incoming cars are strictly smaller than their respective $n_i^{\text{out}} = \mu_i$, namely

$$\langle n_1^{\text{in}} \rangle = 2.5 := \lambda_1 \Rightarrow \frac{\lambda_1}{\mu_1} \simeq 0.83 \qquad \langle n_2^{\text{in}} \rangle = 1.5 := \lambda_2 \Rightarrow \frac{\lambda_2}{\mu_2} = 0.75$$

it is also true that the effective amount of time in which μ_i can let traffic flow is approximately halved w.r.t. the single queue situation without any traffic light. If we assume to deal with something like $\langle \mu_i \rangle \sim \mu_i/2$ we easily see that on average the system is indeed overloaded[7]. Not even Q-Learning is capable of resolving such a scenario: its performance

[7] We may assume that a queue is potentially overloaded if $\frac{\lambda}{\langle \mu \rangle} \geq 1$.

is almost identical to the non-smart case. Apparently, genius drivers do not to make the situation much worse, but that's a tricky point: while producing similar graphs, in the first case traffic slowly flows, while in the second one the most common situation is to experience a completely freezed system. As a final remark, the optimized MDP seems to find a good solution in terms of N. If we look closer, however, we see that its optimal action consists in "shutting down" the shortest queue in order to let $\langle \mu_1 \rangle \sim \mu_1$, and so $\lambda_1/\mu_1 < 1$. Of course, this policy implies $\langle \mu_2 \rangle \sim 0$. While implementing the reward function, one should pay attention to include some sort of fairness metric such as balance between the queues. We'll leave it as an idea for a future work (see Section 7).

5.1 Exploration

Since we have plenty of parameters to play with, here we'll simulate different scenarios for a repeated one hour simulation.

First of all, it is clear that the length of the queues is not, in general, the most critical parameter. To show that, we run the same simulation as before letting both queues to reach $n_1^{\max} = n_2^{\max} = 100$. Results are shown in Figure 3.

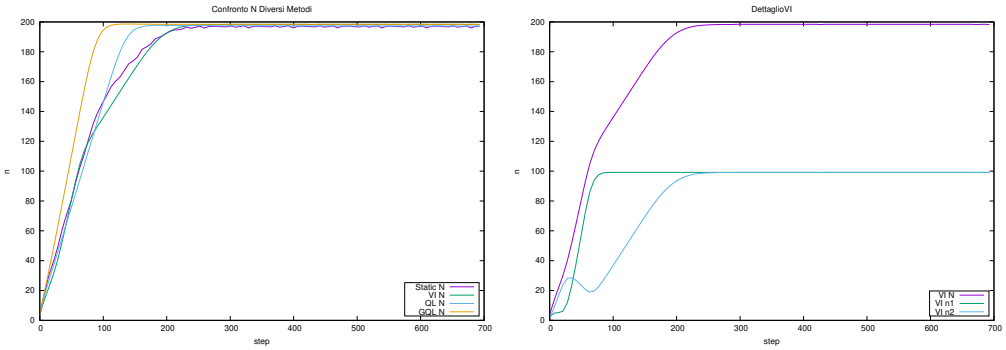

Figure 3: In the same scenario of Figure 2 but with equal, longer queues, not even MDP is able to find a better solution.

Queue length is only intended to absorb fluctuations. Even infinite queues would fill up in a bottleneck scenario. So, w.l.o.g. from now on we fix $n_1^{\max} = n_2^{\max} = 50$.

Now that we showed a totally overloaded scenario, we may ask ourselves what would happen in a critical scenario, i.e. where $\frac{\lambda_i}{\langle \mu_i \rangle} = 1$. For instance, we might choose

$$n_1^{\text{in}} = n_1^{\text{out}} = 5 \qquad n_2^{\text{in}} = n_2^{\text{out}} = 3$$

Results are shown in Figure 4.

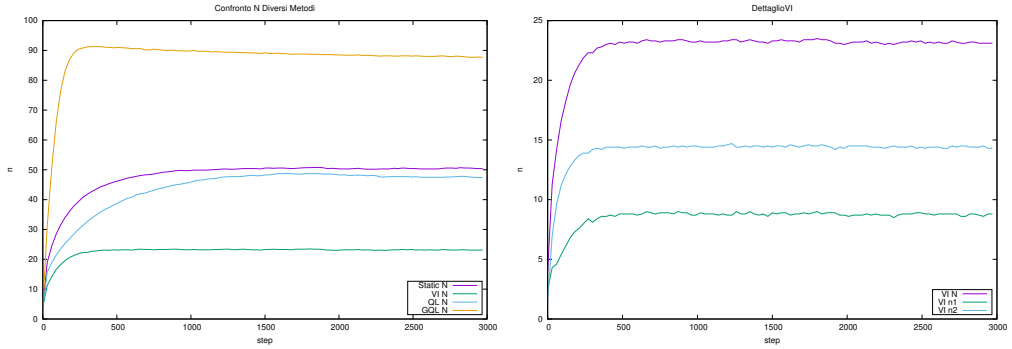

Figure 4: In a critical scenario, the algorithms are capable of finding acceptable solutions. The genius drivers scenario clearly shows that in such a delicate situation bad behaviors lead to almost doubled traffic congestion, no matter the intelligence of the algorithm trying to prevent that.

Here it's interesting to notice a couple of things:

- The Q-Learning algorithm stays somewhere between the optimal MDP solution and the non-smart one. Convergence is more likely to end up closer to the latter.
- We were forced to simulate 4 hours in order to show this slower convergence.
- Despite the perfect symmetry between $\left\{n_1^{\max}, n_2^{\max}\right\}$ and $\left\{\frac{\lambda_1}{\mu_1}, \frac{\lambda_2}{\mu_2}\right\}$, the MDP algorithm seems to have found a "favorite" queue.

We might be tempted to say that congestion in a well-behaving drivers case grows up to half the maximum queue capacity, which happens to be 50. To further investigate whether this is true or not (e.g. it's somehow caused by $n_1^{\text{in}} = n_1^{\text{out}} = 5$) we run the same simulation with $n_1^{\text{in}} = n_1^{\text{out}} = 8$ and another one with $n_1^{\max} = n_1^{\max} = 100$.

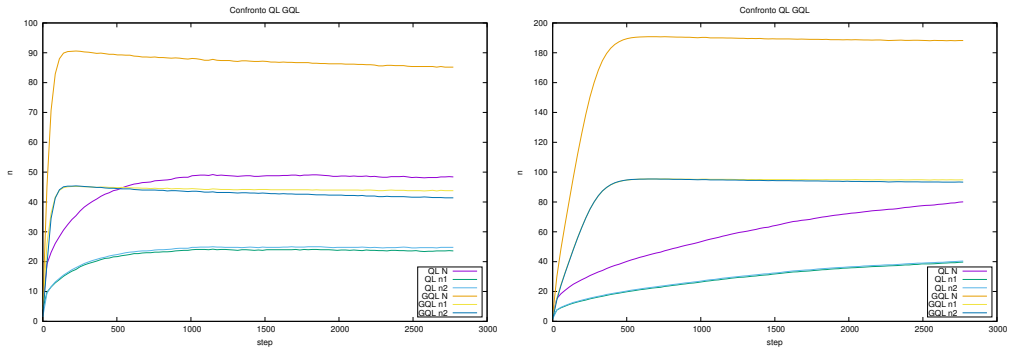

Figure 5: In a critical scenario, congestion is still proportional to queue length.

Despite the fact that in the second case convergence is still quite far, we may still see a dependence on the length of the queues rather than the flux parameters. This was

quite obvious in the overloaded system scenario: $N \to \infty$ means, in practice, to stop at $N^{\max}$.

Our next question is: what if a single queue is overloaded while the other one is not? Of course, the first answer is something like "it depends on how severe the congestion is". In fact, it's easy to imagine that an almost empty queue will easily balance a slightly overloaded one. Similarly, it will be difficult for an almost critical queue to compensate a severely overloaded one. To avoid trivial results, we realize two scenarios:

- Slight "underload", slight overload

$$\left\{ \frac{\lambda_1}{\langle \mu_1 \rangle} = 0.8 \qquad \frac{\lambda_2}{\langle \mu_2 \rangle} = 1.2 \right\} \qquad \Rightarrow \qquad \left\{ n_1^{\text{in}} = 4, n_1^{\text{out}} = 5; \quad n_2^{\text{in}} = 6, n_2^{\text{out}} = 5 \right\}$$

- Severe "underload" and overload

$$\left\{ \frac{\lambda_1}{\langle \mu_1 \rangle} = 0.2 \qquad \frac{\lambda_2}{\langle \mu_2 \rangle} = 1.8 \right\} \qquad \Rightarrow \qquad \left\{ n_1^{\text{in}} = 1, n_1^{\text{out}} = 5; \quad n_2^{\text{in}} = 9, n_2^{\text{out}} = 5 \right\}$$

Since from now on the Q-Learning algorithm will really start to learn something useful[8], we also need to fine-tune the thresholds. Since we enlarged the total capacity of the queues up to $\sim 54\%$, we increase the thresholds accordingly. The new ones will be

$$\texttt{low\_th} = 25, \texttt{mid\_th} = 45$$

Results are shown in Figure 6.

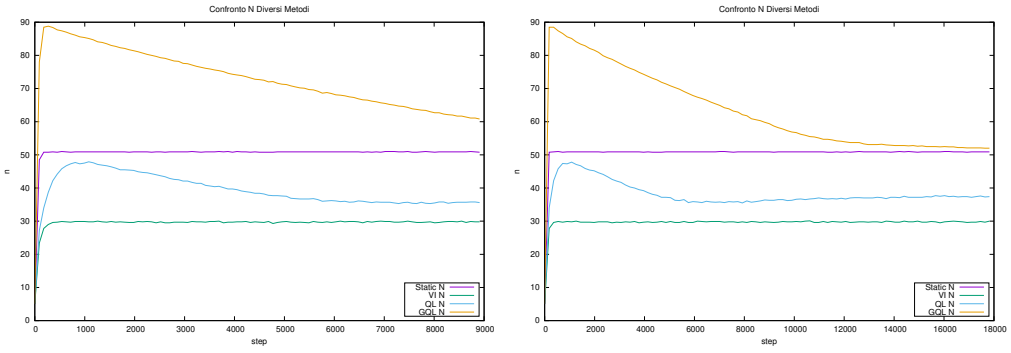

Figure 6: In a slight overload compensation scenario, the Q-Learning model performs well. Convergence for the genius drivers case requires a whole day of simulation.

It is interesting to notice how in this not-so-critical scenario the Q-Learning algorithms dealing with genius drivers obtains almost the same result of the non-smart algorithm dealing with well-behaving drivers. However, here we clearly see that Artificial

[8] i.e. its curve will present a peak, then a decrease, meaning at some point optimal actions are found and begin to be applied while still learning.

Intelligence is not capable of fully compensating human brilliance yet.

The severe congestion case shows a completely different behavior, being more similar to the critical scenario.

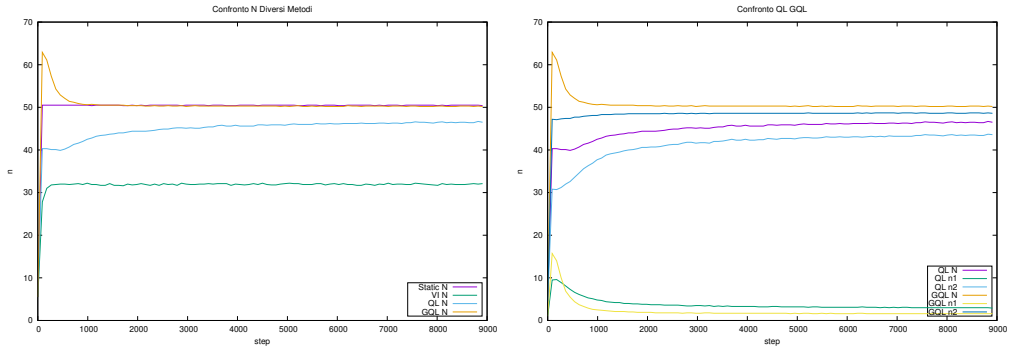

Figure 7: If one queue is severely overloaded, the Reinforcement algorithms learn to unfairly privilege the lighter queue (similarly to the optimal MDP solution in Section 5).

Finally, we may present the ideal case in which neither queue is overloaded.

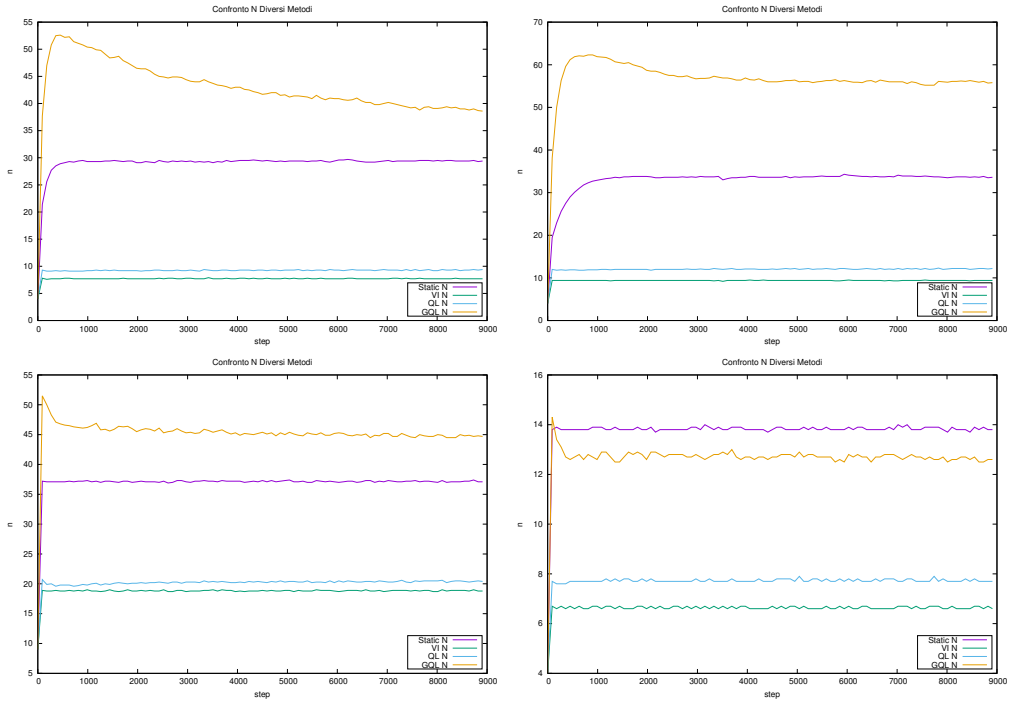

Figure 8: Starting from top left, then clockwise: highest flux queue is critical ($n_1^{\text{in}} = n_1^{\text{out}} = 5$), lowest flux queue is critical ($n_2^{\text{in}} = n_2^{\text{out}} = 3$), both queues are "underloaded" by 20%, both queues are "underloaded" by 10%.

Here, the Q-Learning becomes a good approximation for the optimal MDP algorithm. Congestion reductions become extremely relevant, reaching values up to almost 70% while understanding how to manage a critical, higher flux queue. Also, we can answer to an implicit question: in order to compensate the genius drivers, the load of the network should be at least $\sim 20\%$ under the critical scenario.

5.2 A Dynamic Day

As we have seen in the previous sections, we now know that

- In a total overload scenario, one hour is enough to have convergence.
- In a critical scenario, we might need 4 to 24 hours.
- In a non-overloaded scenario, almost every algorithm stabilizes in less than 4 hours.
- We can divide our day in $\{2, 3, 4, 6\}$ time zones of duration $\{12, 8, 6, 4\}$ hours, respectively.

Let's see how the two Q-Learning algorithms perform in each of these cases[9]. We set as input the critical parameters

$$\left\{ n_1^{\max} = n_2^{\max} = 50 \quad n_1^{\text{in}} = n_1^{\text{out}} = n_2^{\text{in}} = n_2^{\text{out}} = 10 \qquad \texttt{low\_th} = 25,\ \texttt{mid\_th} = 45 \right\}$$

in order to let the algorithm better calculate the percentage variations shown in Figure 1. The reason behind the choice of critical input parameters is the following: each input increase will produce an overload scenario, while each input decrease will produce an "underload" one. Results for the first two cases are shown in Figure 9.

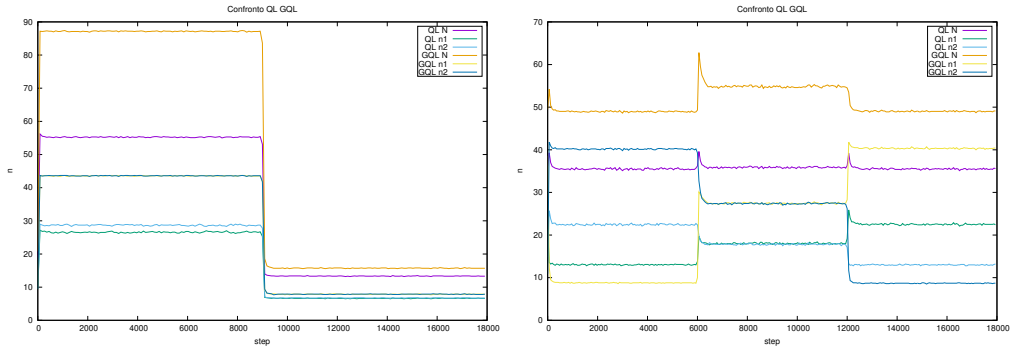

Figure 9: Non-stationary input distributions for `2` and `3` time zones, respectively.

There are multiple remarks already.

[9] Of course, it's pointless to graph the static algorithms as benchmarks.

- In the case `hours = 2`, we clearly see that an abrupt transition between an overload and an "underload" scenario will produce fast convergence. While both algorithms see a decrease in performance in the overload case, it's clear that the `GQL` one suffers it most, speaking of "relative jumps".

- The case `hours = 3` is a bit more interesting. Here, as shown in Figure 1, the goal is to see what happens when the two queues gradually switch their load level[10]. The overall load stays the same, being the sum of the variations equal to 0 for each time zone. Still, we see that while the `QL` algorithm performs always the same we have poorer performances with the `GQL` one in the case where queues are perfectly balanced. We may conclude that it's easier to deal with genius drivers if we're provided with at least one "underloaded" queue.

That being said, we show the last two simulations.

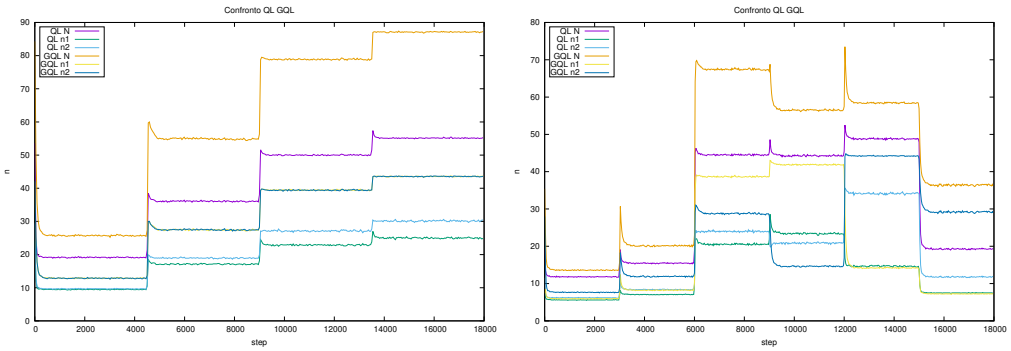

Figure 10: Non-stationary input distributions for `4` and `6` time zones, respectively.

- The case `hours = 4` is intended to explore system response w.r.t. a symmetric, gradual linear increase of traffic load. This behavior is quite expected: jumps gradually reduce their height due to the fact we have an asymptotic limit given by $N^{\max}$.

- The case `hours = 6` is a bit of a random mix of all the preceding scenarios. We present it as an example of a more complex situation.

Of course, all modifications to the load values happen in a abrupt fashion. It is possible to make the discontinuities a bit smoother, but, again, we leave it for Section 7.

[10] It's quite satisfying to check this by following the respective curves over the graph.

6 Results

Here, we briefly summarize the results of the analyses.

First of all, in order to better understand upper and lower performance bounds we implemented two static solutions of the traffic lights problem:

- The "standard", non-smart case takes the switch action every `k` steps. In our simulations, we set `k = 4`, corresponding to approximately $20s$. Being a zero-optimized scenario, we'll assume it to be an upper bound for traffic congestion.
- The over-optimized MDP solution assumes to have a perfect *a priori* knowledge of the system, and it's trained over minimizing N. That's of course a lower bound for traffic congestion.

We correctly showed that the Q-Learning is always somewhere between these two benchmarks, proving that they provide a good reference for the evaluation.

- In severely overloaded scenario (i.e. incoming vehicles are way more than the intersection is able to clear), there's nothing an AI system can do: performances of the Q-Learning algorithm are almost the same as the non-smart case. Even the optimal MDP algorithm only finds unfair solutions, and only when the system is unbalanced (e.g. completely blocking the smallest queue).
- In critical scenarios (i.e. incoming vehicles are exactly as much as the "processing capacity" of the traffic lights) the situation is less catastrophic. We found a big difference between the case where the system is critical because both queues are critical (symmetric) and the one where the system is critical as a result of a combination of an overloaded queue and an "underloaded" one. In the first case, QL solutions tend to be closer to the worst benchmark, while in the asymmetric one we found the most significant learning progression, leading to a performance boost up to 30% (the optimal solution reaches a 40%-ish congestion reduction). However, if this "compensation" scenario is generated by a great difference in traffic load between the two queues, the benefits become less and less visible.
- If neither queue is overloaded (meaning, we have at most one queue which is at most critical), the Q-Learning becomes a good approximation of the optimal MPD solution. We observed congestion reduction up to 70%, but we do not exclude one could fine even better results while playing with the parameters.
- While dealing with bad-behaving drivers, we saw that Q-Learning is particularly good in decreasing traffic congestion when at least one of the two queues is not overloaded nor critical. Unfortunately did not implement the benchmark cases (Section 7), so we're not able to provide fancy percentages, but (with the exception of the severely congested scenario) the learning process is quite evident from the graphs.

- The time zones section is useful to quickly visualize finer behaviors, such as the `GQL` algorithm suffering a bit more in the case of balanced queue lengths.

7 Future Works

If anyone ever wants to further deepen into this rabbit hole, we may leave some ideas to expand the analysis.

- Improving the reward function in order to avoid unfair situations (see Section 5).
- Provide a better model for the `block` dynamic in Section 4.2.
- Provide benchmarks for Section 4.2.
- Model a more complex intersection, or even multiple intersections with multiple queues depending one another. This would also help to write a more realistic logic for the `block` mechanism.
- Provide smoother transitions for the non-stationary distributions analysis.
- Provide a step-by-step live visualization (demo) of the behavior produced by each algorithm.

A Appendix

We've always collected the cumulative reward for each run, but since the graphs were already quite dense (and actually a lot) we decided to ignore it in the report. Here, we'll leave some of the excluded graphs.

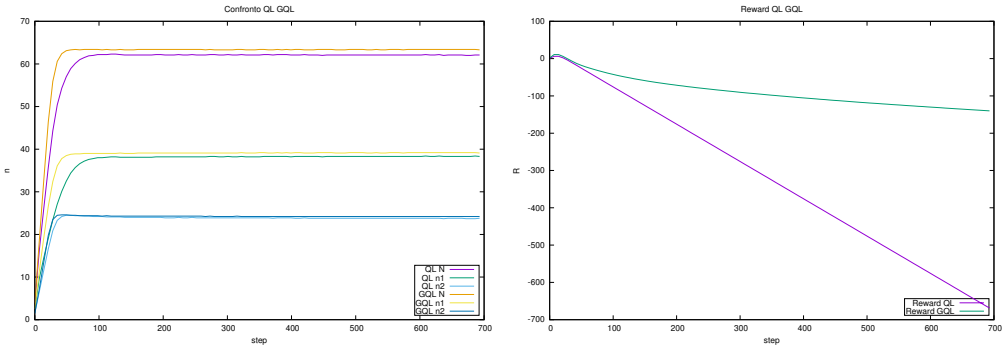

Figure 11: QL and GDL comparisons in N and R for the first case studied in Section 5. GQL seems to perform better because its reward function is defined to strongly penalize (`-2`) `block = 1` and encourages (`+1`) `block = 0`. Since both queues are overloaded, it is sufficient to frequently switch the lights in order to have a positive "block component" reward. Still, of course, the overload of the whole network keeps contributing with negative rewards.

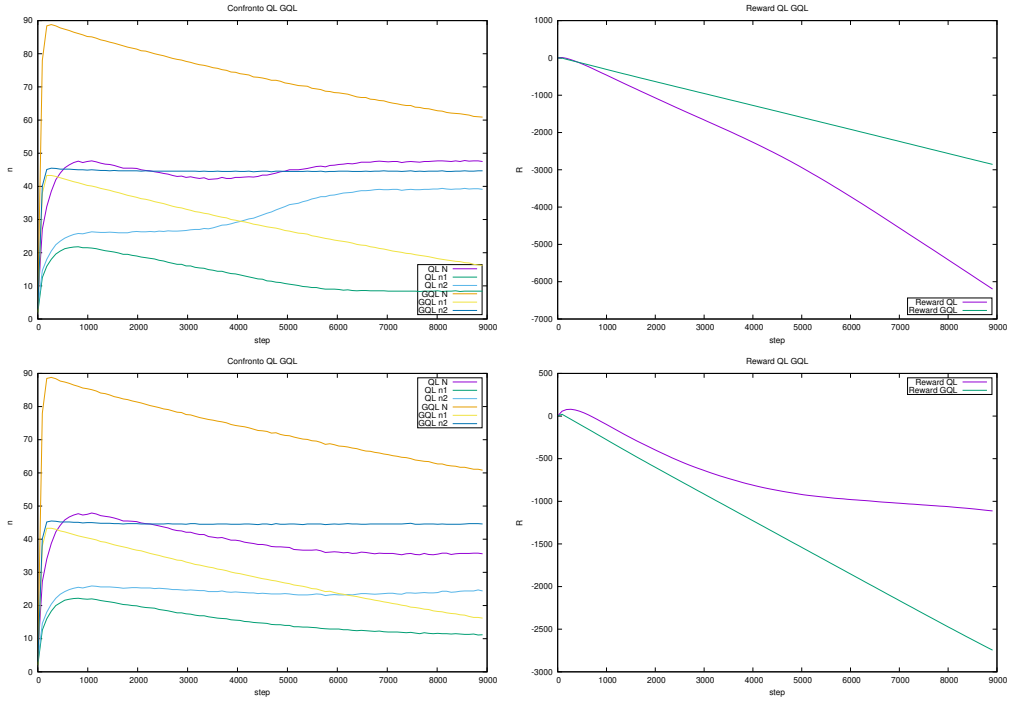

Figure 12: QL and GDL comparisons in N and R for the light compensation scenario (5.1). The upper graphs were produce before updating the thresholds from `(15, 30)` to `(25, 45)`, the lower ones after this operation. Reward curve for QL clearly shows how crucial is to choose adequate values.