Multi-Agent Reinforcement Learning for Traffic Light Control

Hugo Lencastre
Nr 75874
Instituto Superior Técnico
AASMA
Hugo.emanuel.lencastre@gmail.com

Miguel Martins
Nr 81951
Instituto Superior Técnico
AASMA
mfmartins1996@gmail.com

Abstract

In this paper, we test and analyze how we can use Reinforcement Learning (RL) algorithm to increase the efficiency of traffic lights, by reducing the waiting time. To explore RL we used several agents type and different types of databases and several maps to estimate the best agent for each variable. There are three types of Smart or Intelligent Agents, 2 of them more simple that only consider their own intersection and the other that communicates with adjacent intersections to improve traffic flow. The experimental results show that a not basic agent can improve even the most basic scenario even when all the variables tried to help the basic agent.

Categories and Subject Descriptors

Java was the program language chosen, it was used HashMap structures and Circular Fifo Queue structures.

Keywords

Reinforcement Learning (RL), Multi-Agent System (MAS), Q-Learning Algorithm.

1. INTRODUCTION

In traffic control, we are interested in enhancing the safety of all road-users, also improve the flow and reduce the waiting time to the bare minimum possible. To try to achieve these objectives we created several Agents to manage the traffic lights in the intersections in order to improve areas with low congestion and high congestion. Using Simple Agents and Intelligent Agents to control their owns intersections and using the Intelligent Agents with Multi-Agent system (MAS) communication to improve the flow with the adjacent intersections.

Since optimizing traffic can be a complex operation, using Reinforcement Learning (RL) algorithm can be helpful to maximize the flow and can be studied to understanding how they can improve with time and reason about the world.

Motivation.

This idea emerges in the understanding that traffic control is faulty by observing the real-life problem, where we can stay in transit when the road that has a green light doesn't have more cars to move, and as such it appears obvious that the behavior should be dynamic and could be improved.

Problem.

The traffic light control problem consists in multiple nodes, each one an intersection and one node can be connected to another with a road, in our model, the cars are just objects, they not have any intelligence, just drive and choose a destiny. Like in real world, traffic light can't understand what the destination of the car is, so we just optimize the control of the traffic lights and their communication to work the flow. In our problem we only consider one lane in each direction for each road, so when a light change state, all cars can go in any direction except performing a 180º curve (reverse direction).

Assumptions

- There is no detection of collision in interceptions. What this means is that although a vehicle cannot be in the position of another vehicle, it can turn to the same side as one from the opposite direction.
- the 4-traffic light system in each interception only has 2 states, the vertical direction is green and horizontal direction is red and vice-versa, we choose this way because it felt more realistic to the real-world traffic laws.
- Cars don't immediately react, and as such they don't immediately go forward after the car in front stops being stopped. This means that the cars are more realistic in the evacuation after a change to green light.
- The traffic lights in the intersection have a "cooldown", what this means is that the traffic lights have a minimum time before they can change again. We made this choice so the in favor of realism again, the traffic light should not be able to change too quickly.

Simple Agents.

We used three types of Simple Agents, first two change their state after a every regular X steps. The last Agent (SmartAgent) is smarter and he will be the Intelligent Agents competition, he detects how many cars exist in each road that he controls, and if the number of cars in one direction is bigger than on the other direction he changes the state (see more about this state in assumptions).

Intelligent Agent.

So our first Intelligent Agent, uses Q-Learning algorithm [1] to choose when to change the state, from green to red and vice-versa. As can you see further in this paper, the agent in the beginning is very slow and his performance is very bad, but after ten thousand steps it will learn and its efficiency is almost perfect. Each Agent just controls his own intersection and does not communicate with the other Agents if then exist. The qlearning states are based on how many cars exist in each direction, and its actions are thus based on this states. There are 2 actions go vertical green (and horizontal red) or go horizontal green (and vertical red). The reward is the number of cars that passed (as the result of the action)*maxNumberOfCarsWaiting-actualNumberOfCarsWaiting (higher the less cars are waiting), with the exception of the reward being 0 regardless of anything else if a direction has more then 9 cars waiting (road is full).

Intelligent Agent - implementation decisions

Initialy we had the traffic light state as a part of the agents qlearning state, and the actions were do nothing or toggleState, however this had double the states of our current solution for no improvment.

Initially we also had the reward simply being (higher the less cars are waiting), however this would lead to a stabilization of 1 direction being empty being good enough, although there were no more cars needing the green light, therefore we change it to consider the number of cars that would passed as the consequence of the action it took. We also did not have that exception mentioned however it is extremely important in order to create a big backlash in letting a road get full, as it creates big wait times and more importantly it creates traffic in other intersections, which creates chain blockage and ruins other agents learning.

Learning in a Multi-Agent system.

The other Intelligent agent uses a similar approach, but now it communicates with adjacent Agents, for this, every intersection has a Q-Learning algorithm [1] whose state also has incorporated important information given by the adjacent agents, this way the states he learns and therefore the action he chooses are dynamic to the adjacent agents state. This Agent has a similar operation that the previous Agent, in the beginning is not good, but after ten thousand steps it will learn and its efficiency will improve and come close to SmartAgent's efficiency.

Maps.

There are six maps, with one, two, and four intersections. Two maps have one intersection, the difference between both is that the extended can have more cars waiting in the intersection, so the waiting time result can be different, even the simple agent works well. Then we have two maps with four intersections, both similar with the same difference as the basic ones. Here the first two basic agents that use *"StepNr"* can work very well because the cars just can't flow properly. This is the problem in real scenarios, too many intersections and traffic light that just use time to change and do not observe his intersection.

2. Solution

2.1 Dataset

Were made three dataset, simulating 3 hours of the day, rush hour, night hour and daily hour.

2.1.1 Rush Hour

Rush hour have a probability of 1 to spawn a car (highest probability, always spawn when possible), and given a city that has the majority of the traffic coming from the north, was given to North a probability of 0.6, to South a probability of 0.2 and finally from East and West the same probability of 0.1.

2.1.2 Night Hour

Night hour is the most calm dataset, that spawn a car with a probability of 0.2, and because the traffic doesn't have a direction per say, North, South, East and West have a equal probability of 0.25 .

2.1.3 Daily Hour

Daily hour is similar to Nigh hour, but only changes the probability of spawn, that is 0.5. The name of this dataset in the code is *BasicDataSet*.

2.2 Maps

As was previously said, there is five maps, two with one intersection, one map with two intersections and finally two maps with four intersections.

2.2.1 One Cross Map

This map was created thinking in the regions were are less traffic and long roads and sometimes he have a intersection. As was said before, there are two but the only difference is the capacity of cars possible in that road. The name of this map in the code is *BasicMapExtended*.

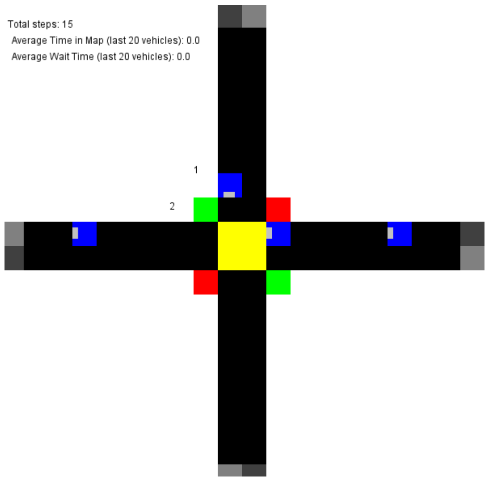

```
Total steps: 15
 Average Time in Map (last 20 vehicles): 0.0
 Average Wait Time (last 20 vehicles): 0.0
```

***Figure* 1.** One Cross Map. The most simple map possible. All Agents can preform great in this case if the Dataset is Night Hour our Daily Hour, if is Rush Hour you can see that the 2 most basic Agent can have some problems.

2.2.2 *Two Cross Map*

The two Cross map was created to experience a little bit of the city, is the middle term between the rural world and city, now you can see the most basic Agents going behind.

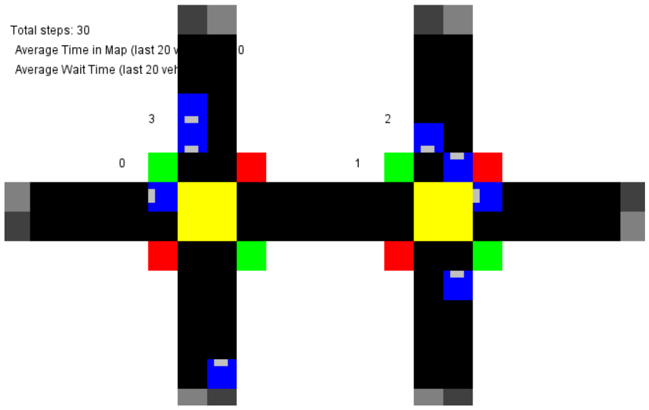

```
Total steps: 30
 Average Time in Map (last 20 v
 Average Wait Time (last 20 veh
```

***Figure* 2.** Two Cross Map. A middle difficulty map, it's already perceptible which Agent can have good results.

2.2.3 *Four Cross Map*

Four Cross Map, the city mode. When you put Rush Hour and this map prepare to see a chaotic map. Basic Agents can't performed well, the Smart Agent has a consistent result, and the Q-Learning Agents, the simple and the Multi-Agent system (MAS) start slow but can be stable and with good performance fast.

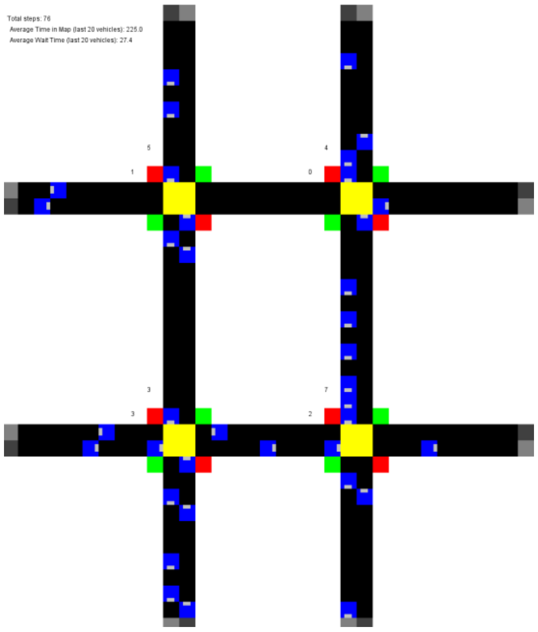

```
Total steps: 76
 Average Time in Map (last 20 vehicles): 225.0
 Average Wait Time (last 20 vehicles): 27.4
```

***Figure* 3.** Four Cross Map. Using to test the infect of the most basic Agents and to proof the Q-learning Agent can learn. This map contains four intersections and eight entrance and exist in the map, witch road have a traffic light that can change only from green to red and vice versa. In total exists four traffic lights in each intersection, and each car can select his destination.

2.3 Agents

There is five Agents, in this point we will

2.3.1 *Basic Agents with "StepNr"*

As was said before, the first Agents switch state every 20 like an ordinary traffic light, *BasicAgent%20*, and the second Agent every 30 Steps, stays in the same state or changes the state 3 times every 10 Steps, *BasicAgent%60*.

```
(1) For each StepNr, between 0 and infinity
        (1.a)if StepNr%60>=30 | | StepNr%20 = =0
                (1.a.1)Return True
        (1.b)Return False
```

2.3.2 *Basic Agent that can be Smart*

This "Smart" Agent just can know how many cars has in each road, so if count 2 cars more in one road that in the other, it changes the state.

```
(1) For each Step, between 0 and infinity
    (1.a) vDirNr = Nº Cars in Vertical Roads in his intersection
    (1.b) hDirNr = Nº Cars in Horizontal Roads in his intersection
    (1.c) If vDirNr > = hDirNr then
        (1.c.1) Return True
    (1.d) Else
        (1.d.1) Return False
```

2.3.3 *Q-Learning Agent without communication*

The Q-Learning Agent is made with Q-Learning [1], to work we created two Hash Maps to guard the Map of each agent and the other to guard the "*LastAction"* made by that Agent.

The *Reward* is calculated and the Q-Values updated when we know the outcome of the action, which is in the next step when we consider the next action to take.

```
(1) New HashMap<CrossRoad,QAgent> AgentMap
(2) New HashMap<CrossRoad,Action> LastAction
(3) For each Step, between 0 and infinity
    (3.a) nrHV = Nº Cars in Vertical Roads in his intersection
    (3.b) nrVV = Nº Cars in Horizontal Roads in his intersection
    (3.c) If AgentMap not contains CrossRoad
        (3.c.1) Create QAgent
        (3.c.2) Start QAgent
        (3.c.3) Put QAgent in AgentMap
        (3.c.4) Action = 0 and put in LastAction
        (3.c.5) Return False
    (3.d) Else
        (3.d.1) Retrieved QAgent from AgentMap
    (3.e) Get NewState
    (3.f) Get Reward
    (3.g) Update QAgent with NewState and Reward
    (3.h) Retrieved the action that QAgent decided to do
    (3.i) Replace Action in LastAction
    (3.j) If Action equals 0
        (3.j.1) Return False
    (3.k) Else
        (3.k.1) Return True
```

2.3.4 *Multi-Agent System (MAS) with Q-Learning Agents*

The main difference from this Agent to the simple Q-Learning Agent is that in the function *GetState*, the calculation for the *NextState* no longer consider only his intersection but the *NearIntersections* also.

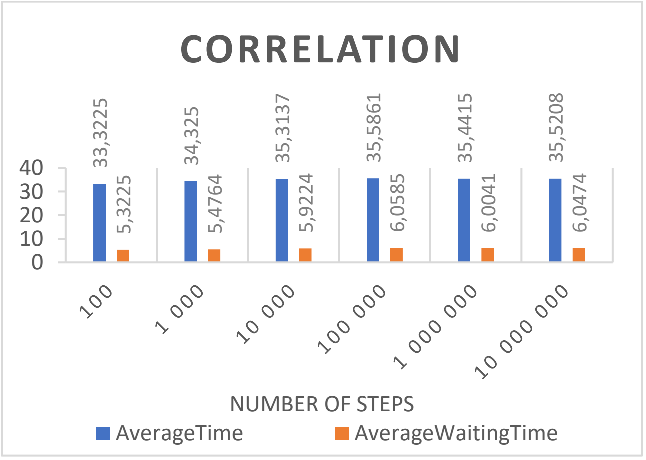
```
(1) GetState (CrossRoad cs, Integer nrHV, Integer nrVV)
    (1.a) Get csNearList = List of adjacent CrossRoads
    (1.b) NewState = nrHV + nrHV*10
    (1.c) For i between 0 and length of csNearList
        (1.c.1) NewState +=csNearList.get(i) * 10^(2+i)
    (1.d) Return NewState
```

3. Results

After run all the test, exist a correlation between *AverageTime* and *AverageWaitingTime*, to show this correlation the first graphic will be that prof.

3.1 Correlation

The reason why steps, one hundred to ten million, are important is because, until step one hundred there's almost no traffic arrive destination, so no statics available, so in step one hundred that the first sequence of cars arrive. Because of that the waiting time in this step is the lowest, most of the time. Then we can say that in the most cases, in step ten million, waiting time converged to the value that is represented in that step.



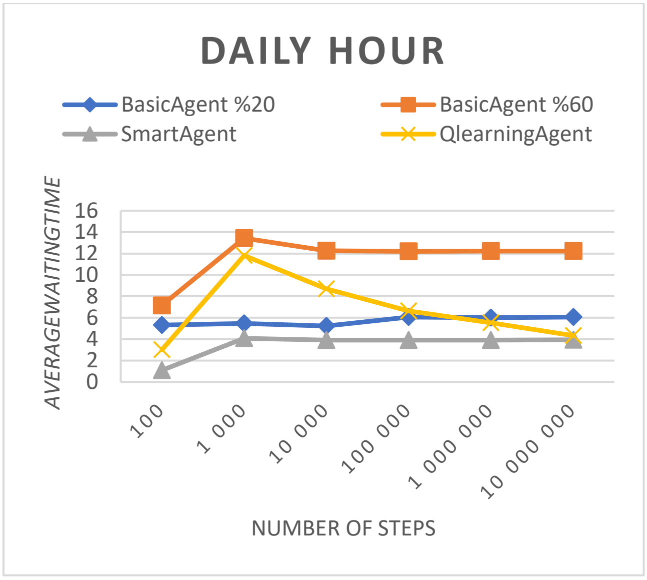
3.1 One Cross Map Extended

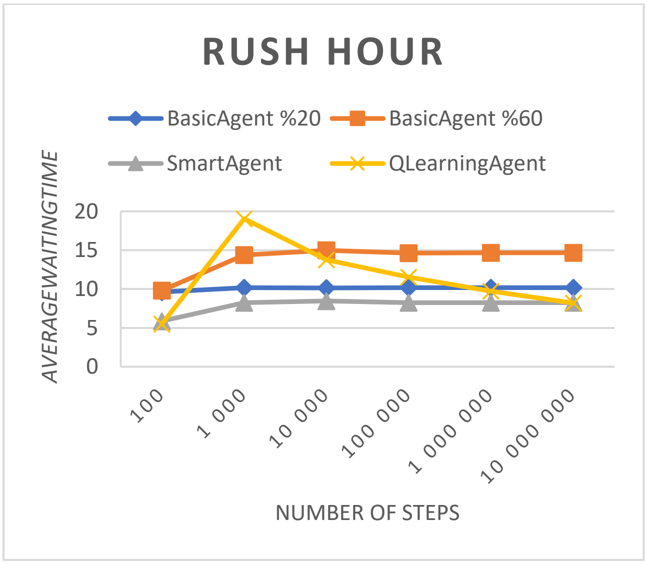
In One Cross Map the AverageWaitingTime is between 1 and15. The best Agent is the SmartAgent with the low AverageWaitingTime equal to 1,0882 but when it converge, the lowest AverateWaitingTime is equal to 3,9120 at step one million, but a very stable Agent, with 4 steps with similar values near 3,9.

QlearningAgent has a steep climb to AverateWaitingTime equal to 11,8072 at step one thousand but the best result is equal to 4,3248 at step ten million. Because this Agent doesn't know what to do until it experiments, and then start to learn and improve their performance.

The two basic agents, the BasicAgent%20 and BasicAgent%60 have a similar difference between their AverageWaitingTimes, the difference is near to 6. The BasicAgent%60 is more slow because in the beginning it waits thirty steps.

In rush hour there is a similar behavior but in this case the QLearningAgent as a higher peek, with AverageWaitingTime equal to 19,0468 and a lowest of 8,2150 that is better than SmartAgent. We can say that is the only case that QLearningAgent wins to SmartAgent.

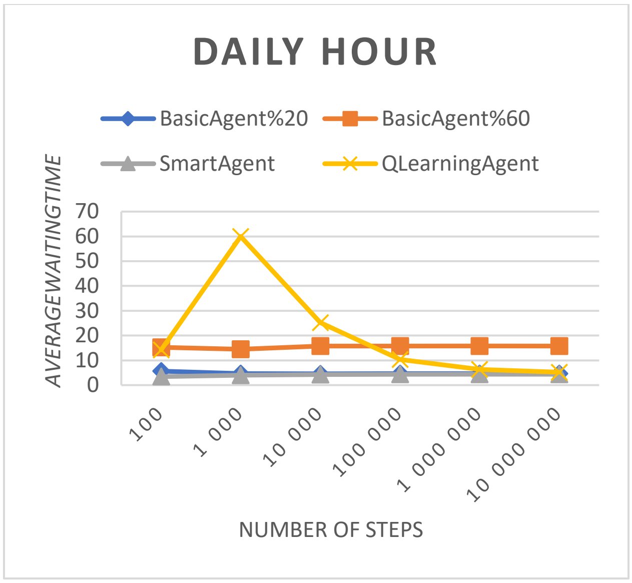


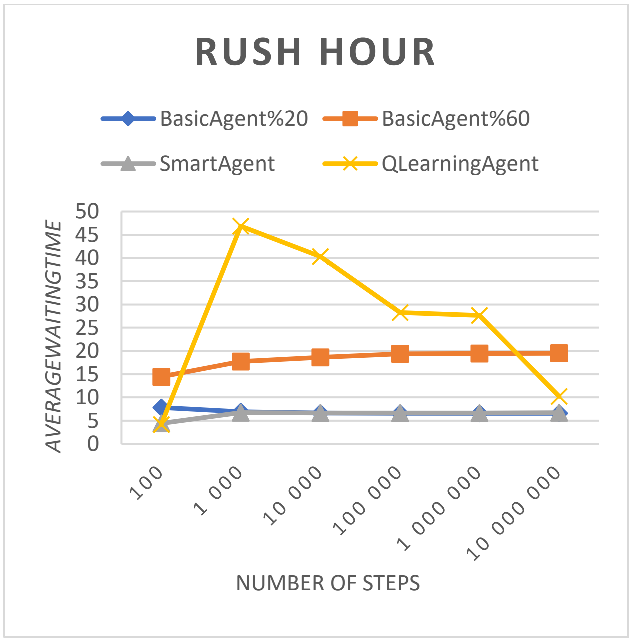


3.2 Two Cross Map Extended

In this middle map, we can see that *QLearningAgent* can be disaster until it learns and stabilize near the performance of *SmartAgent*,at near 4,5 for the AverageWaitingTime, in daily hour, but can't catch it. In rush hour the difference is 4, the SmartAgent as a value of 6,67 and QLearning as a value of 10,19.

The two basic agents, the BasicAgent%20 and BasicAgent%60 have a similar difference, about 9, both for daily and rush hour.





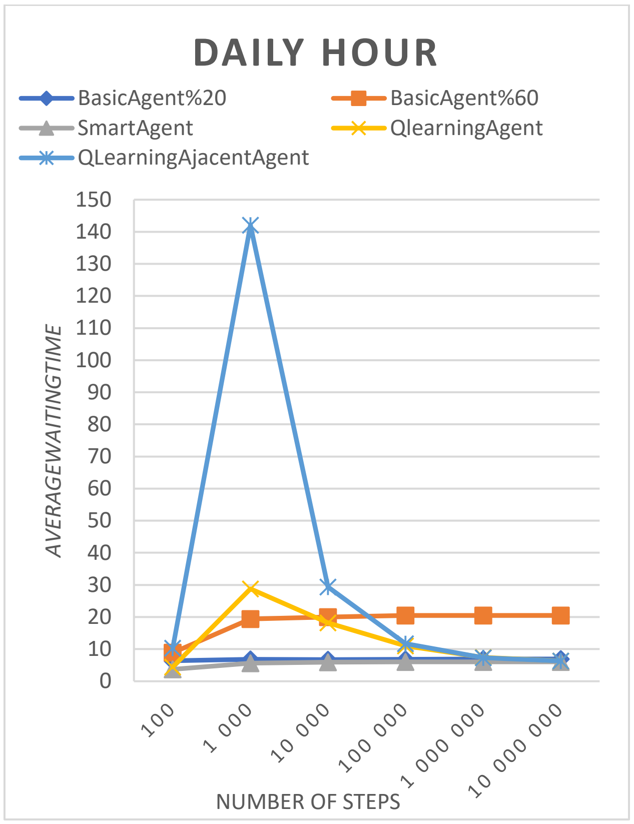
3.3 Four Cross Map Extended

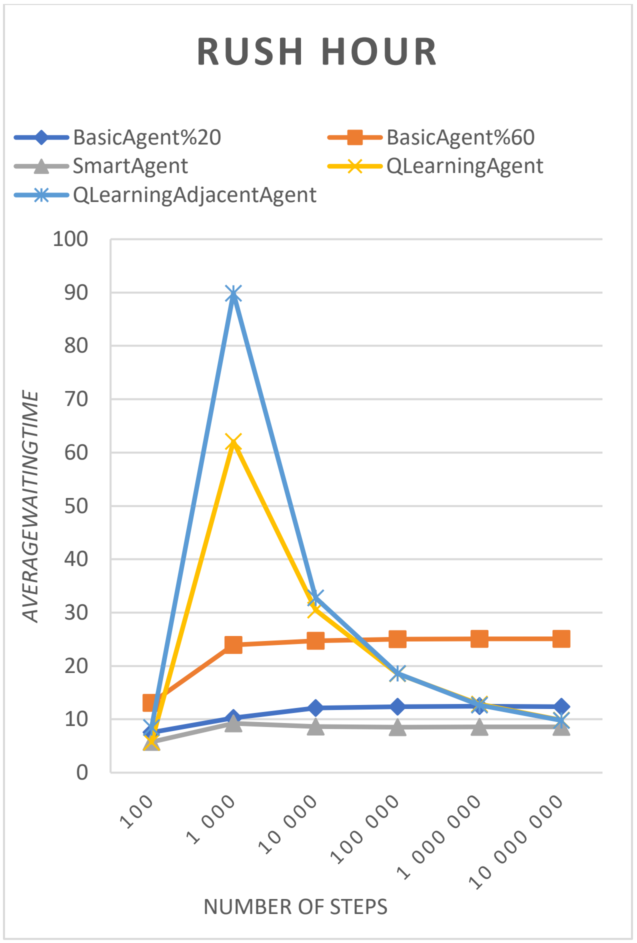
In Four Cross Map, we can say that if QLearning was a disaster in Two Cross Map, here is astronomic, but also is astronomic the comeback. The peak in rush hour is 62,0663, in daily hour is 28,7852. But in the end he as a result of 9,814 just a one value up from SmartAgent in rush hour and value of 0.2 in daily hour.

In this map we introduce QLearningAdjacentAgent that as a very poor performance, in the beginning, in both hours, with a peek of 316,0955 in daily hour and a peed of 219,0876 in rush hour. The lowest number is very surprising, even lower that the QLearning in step one million, value of 12,6895, and ten million, value of 9,7907, in rush hour, a very similar value in daily hour.

SmartAgent has a very stable performance again, with values around 8, from step ten thousand to ten million.

Basic agents have similar behaviors like before but with values a bit higher. But with the difference similar as before.





4. CONCLUSION AND LAST NOTES

The basic agent do not perform as well as the other agents, at higher steps the Intelligent Agents seems to have similar behavior to the SmartAgent, something they learnt by themselves, which is invaluable because circumstance can easily change and the Intelligent Agents have the capability to adapt.

Other interesting circumstances that the Intelligent Agents would be able to learn are:

- Traffic that changes by the time of the day.
- Traffic that changes because of an event.
- More complex intersections and maps.

These circumstances could be learnt from by changing the state and reward of the agent, but a good policy for the SmartAgent might not be so direct as in our simple examples compared to the real-world.

5. REFERENCES

[1] Xianshun Chen, Q-Learing algorithm, Seattle, WA, USA, https://github.com/chen0040/java-reinforcement-learning