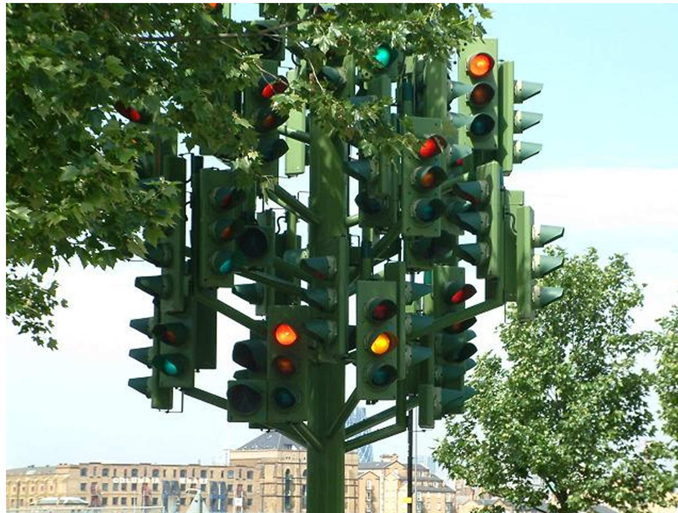
Reinforcement Learning of Traffic Light Controllers under Partial Observability

MSc Thesis of
R. Schouten(0010774), M. Steingröver(0043826)



Students of
Artificial Intelligence
on
Faculty of Science
University of Amsterdam, The Netherlands

Supervisor Dr. Bram Bakker

August, 2007

Abstract

Due to the increasing amounts of traffic in and around urban areas there is a growing need for intelligent traffic lights that optimize traffic flow. This thesis attempts to implement more realism in intelligent traffic light control. In a realistic traffic light control system there is a lot of partial observability. This thesis provides a description and solution to optimize and adapt traffic light controllers that use multi-agent model-based reinforcement learning algorithms to now support partial observability. Multiple methods are described in this thesis. Also a way of learning the model using partial observed data has been implemented. This thesis will show that the methods used to cope with partial observability and even the methods implemented with which to learn the model under partial observability obtain very good results and even outperform the original reinforcement learning algorithms in some cases.

Contents

1 Introduction **5**

2 Markov Decision Processes **8**
- 2.1 Value Iteration . . . 10
- 2.2 Reinforcement Learning . . . 11

3 Partially Observable Markov Decision Processes **14**
- 3.1 Beliefstates . . . 16
  - 3.1.1 Bayes' rule . . . 16
- 3.2 Value Iteration for POMDPs . . . 16
- 3.3 Heuristic Approximation for POMDPs . . . 19
  - 3.3.1 Most Likely State . . . 19
  - 3.3.2 Q-MDP . . . 20

4 Green Light District Simulator **21**
- 4.1 State Representation . . . 21

5 MDP Implementations of Reinforcement Learning for Traffic Light Control **24**
- 5.1 Standard . . . 24
- 5.2 Congestion . . . 25

6 POMDP Implementations of Reinforcement Learning for Traffic Light Control **27**
- 6.1 Sensor noise . . . 27
- 6.2 Incomplete sensor information . . . 27
- 6.3 State Estimation . . . 28
  - 6.3.1 Implementation . . . 28
- 6.4 Decision Making . . . 29
  - 6.4.1 All in Front implementation . . . 31
  - 6.4.2 Most Likely State implementations . . . 31
  - 6.4.3 Q-MDP implementation . . . 32
  - 6.4.4 Beliefstate Approximation . . . 33
- 6.5 Learning the Model . . . 35

7 Test Environment and Statistics **36**
- 7.1 Test Domains . . . 36
- 7.2 Statistics . . . 38
  - 7.2.1 Average Junction Waiting Time . . . 38
  - 7.2.2 Total Arrived Road Users . . . 38
- 7.3 Different Set-ups . . . 38

8 Experiments & Results **40**
8.1 Experiment 1: COMDP vs. POMDP algorithms 41
8.2 Experiment 2: Learning the model with MLQ vs. COMDP . 43
8.3 Experiments 3 & 4: Sensor Errors 45
8.4 Experiment 5: Gaussian vs Uniform road user noise methods 47
8.5 Experiment 6: Explaining COMDP performance 48
8.6 Experiment 7: Improvements for COMDP? 50

9 Conclusion & Future work **53**

References **55**

A Work distribution **57**
A.1 Written sections 57
A.2 Programmed parts 57

1 Introduction

The increasing amount of traffic in the cities has a large impact on the congestion and the time it takes to reach a certain destination. But not only the amount of traffic but also how you deal with this traffic has a large impact. Adding roads is not sufficient by itself, since they will always reach an end point, like bottlenecks or junction. Bottlenecks cannot be prevented. However the way junctions are controlled has a lot of room for improvement. If junctions are controlled, it is mostly done by traffic lights.

Traffic lights though, are most of the time not adaptive. The classic traffic light controller has a 'fixed-cycle' which does not take in account how much traffic comes from any direction, it just switches configurations of lights on a timer interval. This often causes road users to wait at a completely empty junction with only one road user waiting for a red sign. Improvements already have been made, by putting sensors in the lanes in front of the traffic lights to let the controller only cycle between occupied lanes, thus disabling the chance of having to wait at a red light at an empty junction. An extension to this are "green waves", which are multiple traffic light controllers linked together. Subsequent traffic lights in the "green wave" are programmed to switch to green signs in a wave like fashion. So when road users are detected at the first traffic light, they will be able to get green signs for all the next junction participating in the "green wave".

More theoretical approaches to improve the traffic light control include machine learning algorithms, which adapt to situations. Machine Learning algorithms store the sensor information the sensors gather about the road users crossing the junction. These stored sensor information samples provide a way to predict the future driving behaviour of road users and therefore enable the traffic light controller to calculate future waiting times for those road users for each actions the traffic light controller can make. When the controller has the actions combined with waiting times, the optimal action would be to do one of these actions where the expected waiting times are the lowest.

In this thesis a Reinforcement Learning implementation is used on a simulation of a simplified city with roads and junctions. The simulation has the possibility to get all the information that is needed to make a decision and is therefore *completely observable*. Viability of these machine learning techniques is shown in [21, 22, 15]. These theoretically proven methods do not take some issues into account, like partial observability. In the real world it is not realistic to assume that all the information can be gathered or that the gathered information is 100% accurate. Thus decisions have to be made based on incomplete or erroneous information. This implies working with *partial observability*.

The used simulator has a discreet grid-like state space representation of the whereabouts of all road users. Running the simulator generates data;

this data exists of road users moving from one grid point to another. Every road user is in a state where the action, turning the traffic light to red or green, effects the transition of the road user to another state in the next time step, this is called the state transition. When counting state transitions the probability distributions can be created which are used to form the fully observable model [21, 22] or in other words the completely observable model. Completely observable means that the controller has complete access to the state space.

In the real world a similar grid like abstraction can be made on roads. A direct mapping can be created by putting sensors on every grid point on that road. This would be the equivalent for the completely observable simulation. On a real road the amount of sensors needed to realise this is immense, moreover road users generally do not move from one grid point to another, but move continuously. Approximating the state space by using less sensors than grid points convert the problem from completely to partially observable. The controller has no longer direct access to the information of the state. Less sensors make the problem harder, due to lack of data. The quality of the sensors also play a role in partial observability. To be able to use reinforcement learning, as is done on the completely observable grid, on the real world roads there is need for a way to handle partial observability. The implementation of the partially observability in the simulator will consist of at most two sensors.

This thesis is about partial observability and solving the partial observability problem in the real world. To be able to test and make conclusions it is imperative that simulator is able to model and handle partial observability. To model partial observability in the – up to now fully observable – traffic simulator an observation layer is added. This allows algorithms to get their state information through this observation layer instead. In a real world application the controller would have to be implemented similarly, by getting its state information from observing the movement of the road users.

Multi Agent Systems with implemented partial observability is a domain not heavily explored and important to make a link to the real world application and realistic traffic light control. It is therefore a highly interesting domain. The implementations are not straight-forward, because many of the already developed algorithms to solve partial observability are centered around single agent problems. Another problem is that many are too computationally intensive to solve problems more complex than toy problems. Adding the Multi Agent approach increases the computation complexity. Instead having to compute a best action based on one agent, the combined best action needs to be chosen from all the combinations of best actions of the single agents.

To diminish the computational complexity several heuristic approximations are possible in solving partial observability. Examples of heuristic approximations are Most Likely State(MLS) and Q-MDP[1].

The thesis has the following set-up:

- The next section gives an overview of the Markov Decision Process and all concerned basics.
- In Section 3 there will be an overview of POMDP basics.
- Section 4 will show the Simulator used.
- Section 5 will give the MDP implementations done in the past and other done research.
- After that, in section 6 an overview is presented of all implemented POMDP methods and algorithms and state estimation and decision making.
- Section 7 describes the performed experiments and the statistical tools needed to compare the algorithms.
- In section 8 the results will be given.
- And at last in section 9 there will be the conclusion, discussion and future work possibilities.

2 Markov Decision Processes

A Markov Decision Process or MDP, a class of problems, gives a mathematical framework which can be used for decision making under randomness or uncertainty. A MDP is a tuple $\Xi = (S, A, R, T)$, where S is a finite set of states $S = \{s_0, s_1, ..., s_n\}$ which cannot be controlled directly, A a finite set of actions $A = \{a_0, a_1, ..., a_n\}$ that an agent can perform and therefore can be controlled directly. T is a stochastic state transition function $T : S \times A \rightarrow \Pi(S)$. The notation for the probabilities are $P(s^{t+1} = s'|s^t = s, a^t = a) = \tau(s, a, s')$ which are the probabilities that an agent will get to state s' when he's currently located in state s and performs action a. R is the expected real-valued reward $r : S \times A \times S \rightarrow \mathbb{R}$ function, which gives the expected reward. This is acquired by performing an action a that enables the transition $s \rightarrow s'$.

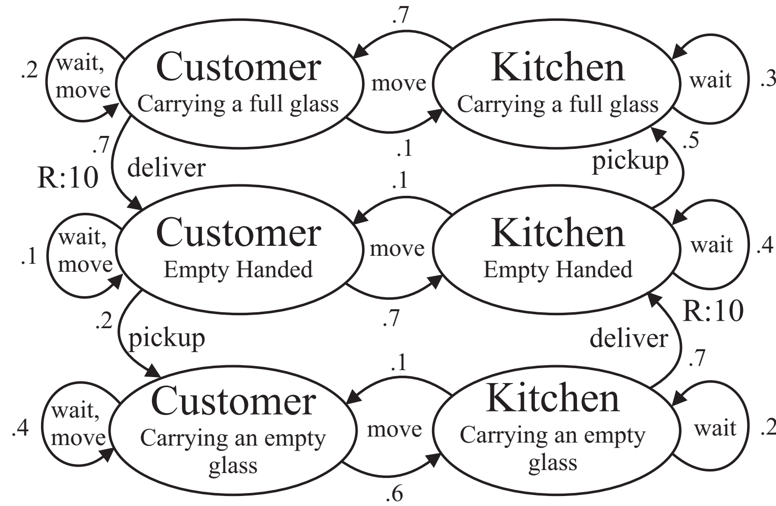


Figure 1: The MDP model for the serving bot example.

Example. *The MDP model diagram of a serving bot is shown as an example in figure 1, to give a slight intuition on the above definition of the MDP. First all of the elements of the MDP model Ξ are described:*

S: There are six states s in S, these are: {'Customer, carrying a full glass', 'Kitchen, carrying a full glass', 'Customer, empty handed', 'Kitchen, empty handed' 'Customer, carrying an empty glass', 'Kitchen, carrying an empty glass'} The serving bot is at one of these states at all times.

A: The different actions the serving bot can do are: wait, the bot does nothing and will therefore remain in its state from which it executed the wait action. Move: a physical motion move of the robot, moving itself physically from one location to another. It can move from a

customer to another customer, from a customer to the kitchen or vice versa. Deliver: the robot can deliver an empty glass in the kitchen or it can deliver a full glass to a customer. In both cases the robot will be left empty handed. The last action the robot can do is 'pickup': to pick up an empty glass from the customer, leaving the robot carrying an empty glass. Or it can pick up a full glass at the kitchen, getting it to the carrying a full glass state.

R: To make the robot worthwhile for the robot, it needs to get some kind of reward now and then. When the robot delivers an empty glass at the kitchen or a full glass at a customer, it will get 10 point reward. Defining it as the reward function r: r(s,a,s') = r('CustomerFullGlass', 'deliver', 'CustomerEmptyHanded') = 10 and r('KitchenEmptyGlass', 'deliver', 'KitchenEmptyHanded') = 10. Al the other combinations r(s,a,s') will be 0.

T: The robustness of a MDP is its ability to act under uncertainty, therefore the stochastic transition $T : \tau(s, a, s')$ *adds a factor of uncertainty to state transitions. All outgoing actions from one state form a probability distribution, together summing up to 1. In figure 1 the transition probabilities are shown as the fractions. For example when the robot is in the state 'Customer, empty handed' the robot will have a .2 probability that there will be an empty glass at the customer which it can pick up. In .7 of the situations the robot would go back to the kitchen and .1 of the situations lead to staying at the table or moving to another customer.*

A fundamental property of the Markov Decision Process is that the states hold to the Markov property. This allows the decision maker to make its decision only based on the state the agent is currently in, one would not need to take the entire history into account to be able to know what the follow up states could be. Thus the Markov property is expressed in the following formula:

$$\Pr(s_{t+1} = s | s_t = s_t, \ldots, s_1 = s_1) = \Pr(s_{t+1} = s | s_t = s_t) \tag{1}$$

The task of an agent is to determine a policy $\pi : S \rightarrow A$. At each discrete time step t the agent selects an action a_t given the state s_t, that maximizes the cumulative future discounted reward or return R:

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ... = \sum_i \gamma^i r_{t+i} \tag{2}$$

where r_t is the reward at time t, and γ is a factor between 0 and 1 which discounts rewards further in the future, letting the rewards further away in the future be less important. The more γ is closer to 0, the more short term rewards are favoured over long term rewards.

2.1 Value Iteration

It is important to know how good a state or, by performing an action, a potential future state is for the agent, this can be calculated by use of a value function. The return described above is the expected value which shows how good the action or state really is. The expected value for a state can be expressed recursively using the return:

$$\begin{aligned} V(s_t) = & \quad E\left\{\sum_{i=0}^{n} \gamma^i r_{t+i}\right\} \\ = & \quad E\left\{r_t + \gamma \sum_{i=1}^{n} r_{t+i}\right\} \\ = & \quad E\left\{r_t + \gamma V(s_{t+1})\right\} \end{aligned} \tag{3}$$

The values calculation in equation 3 is done in a recursive way, this has a drawback that it can be hard to know at which point to stop the recursion. In order to not have this problem the values can also be calculated iteratively with n update steps. This is shown in the following equation:

$$V_{k+1}(s) \leftarrow \sum_{s'} \tau(s,a,s')(r(s,a,s') + \gamma V_k(s')) \tag{4}$$

Value Iteration initializes all values in V_0 at some value, in most cases this will be zero, but it can also be set to some random value. In this thesis all initialization values will be set to zero. For each iteration k, all the values in the vector V_{k+1} are updated from the values stored in V_k as described in function 4. Intuitively the first iteration $k = 0$, all values that are stored in V_1 consist of direct rewards only. The next iteration will define the Values in V_2 being the direct rewards added to the discounted direct reward of the next step. Therefore the lookahead range for the rewards is increased by one state transition every iteration. When there are loops in the process, endless follow-up states can be found. However there is still a way to know when enough iteration have been calculated. Namely by comparing the difference between the current and previous value of the same state: $\Delta V(s) = |V_{k+1}(s) - V_k(s)| < \theta$. Where θ is a threshold close to zero.

When the threshold has been reached the assumption can be made that the learning has converged and that the optimal values V^* have been calculated.

The optimal policy will then be

$$\pi^* = \underset{a}{\operatorname{argmax}} \, [r(s,a,s') + V^*(s')] \tag{5}$$

Taking the action that maximizes the future reward of getting into that next state together with the reward of getting to the state.

Example. *With value iteration, for each of the states in the serving bot example the values can be calculated. Iterating roughly 12 times will yield the values that are shown in figure 2. The optimal action for the serving*

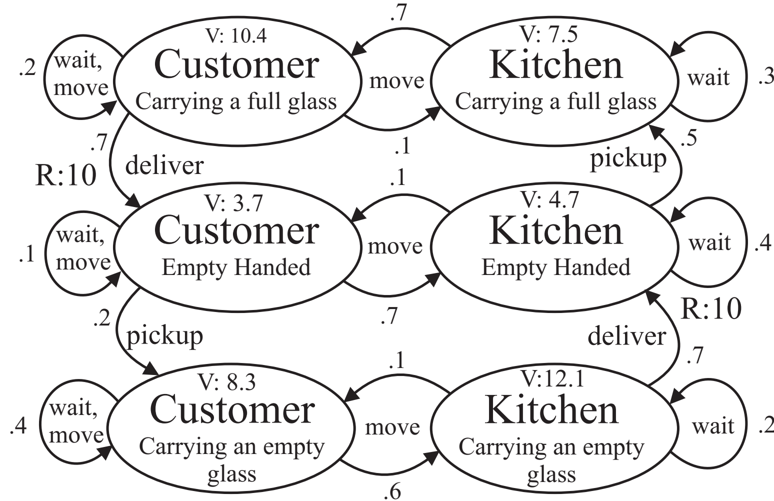


Figure 2: The MDP model for the serving bot example with the added Values for each state after a dozen value iterations. A γ of 0.8 is used.

bot in for example state 'Customer, Empty Handed' can be calculated using equation 5. The states it can reach from 'Customer, Empty Handed' are 'Kitchen, Empty Handed', 'Customer, carrying an empty glass' or staying in 'Customer, Empty Handed'. While there are no direct rewards in this state, the reward of reaching one of those next states is simple the discounted value for that state. The rewards for getting to the next states are respectively: 4.7, 8.3 and 3.7. If the robot is able, cause there need to be an empty glass present, it will take as its best action 'Pickup', to get to state 'Customer, carrying an empty glass', otherwise if there is no empty glass it will return to the Kitchen as that has its second highest expected reward.

2.2 Reinforcement Learning

Its not always the case that the transition probabilities are fully known or known at all. When the transition probabilities are not known a dynamic programming approach such as Value Iteration cannot be used, since $\tau(s, a, s')$ is an integral part of equation 4.

The sub MDP solving class where these kind of problems are approached is called reinforcement learning. Reinforcement learning has been applied to various domains with considerable success. One of the most impressive results so far is in backgammon[17], where a program based on reinforcement learning is able to play world-class expert level. Reinforcement Learning is a technique for learning control strategies for autonomous agents from trial and error[4, 13]. The agents interact with the environment by trying out actions, see figure 3, and use resulting feedback (reward and the consecutive state) to reinforce behaviour that leads to desired outcomes. Instead of

calculating the values iteratively prior to the start of the process as was done in equation 4, the values are updated after each action an agent performs.

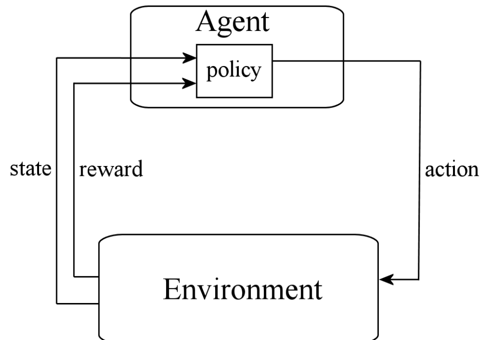


Figure 3: The generic model of an agent doing actions in its environment while getting the information about the state its in and the reward corresponding to the action and the state

Learning the environment by trial and error can be done in several ways. One way is the direct approach, where the agent finds its rewards during exploration of the environment and add them directly to previous found rewards. This is called Q learning[20]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{6}$$

Here $r(s,a)$ is the direct reward the agent gets when doing an action a in state s and the γ is the discount factor for future rewards. It is intuitive to see that rewards in the future are not as important as the immediate rewards. Q learning is closely related to value iteration, the Q-value action pairs are updated in the same iterative fashion.

Another way of dealing with the fact that the state transition probabilities are not priorly known, is to learn the state transition probabilities. This is done by counting state transitions as the agent execute them. State transitions form the model of the process. This is model based reinforcement learning, which is also an approximation to the dynamic programming approach, Value iteration. An example is Sutton's Dyna[14].

The equation for the model based reinforcement learning.

$$Q(s,a) \leftarrow \sum_{s'} \tau(s,a,s')(r(s,a,s') + \gamma \max_{a'} Q(s',a')) \tag{7}$$

Example. *Every time the serving bot comes to a state it adds one to the counter of getting to that state, the probability is then the value of the counter of going to that state divided by the values of the counters of all transitions*

leading from that state. Lets say that the state 'Customer, Empty Handed' has been visited 10 times. The transition probabilities suggest then that of those 10 times, two of them had a customer waiting with an empty glass, causing the bot to pick it up. Six of the times the bot went to the kitchen and one time it stayed at the Customer. If the bot would encounter a customer having an empty glass at his eleventh time it visited that state, the probabilities would be different, and the Value functions would not be good any more. With Reinforcement learning, every time the agent moves equation 7 is used one more time with the updated $\tau(s', a, s'')$.

3 Partially Observable Markov Decision Processes

Partially Observable Markov Decision Processes, or POMDPs for short, are a set of processes in which the controllers, make decisions using partially observable Markov states, this in contrast to the previously mentioned MDPs, where the states are completely observable. Completely observable MDPs (or COMDPs) are the same as normal MDPs, except that it is used in this paper to make a more noticeable difference between partial and completely observable MDPs. COMDPs and POMDPs both control the actions of an agent to move from one state to another and gaining a reward for that action. Unlike in COMDPs, where the state in which the agent is in is directly accessible by the decision maker, POMDPs will model a controller that will have to do with only the observation of being in a state. In the end the observations will lead to a beliefstate which is a probability distribution over the state space.

A POMDP is a more general model than a COMDP, which can handle tremendous more problems in the real world, since most of the time the real state is not really directly accessible by the decision maker. The POMDP is denoted by the tuple $\Xi = (S, A, Z, R, T, O)$, where S,A,T are the same as in the COMDP tuple. Z is the set of observations. After a state transition one of these observations is perceived by the agent. O is the observation function, $O : S \times A \rightarrow \Pi(Z)$, mapping states-action pairs to a probability distribution over the observations the agent will see after doing that action from that state. As can be seen, the observation function is dependent on the state the agents is in and the action it executes, it will not be entirely certain which state it will reach although it has an estimation. The general reward function, $R : S \times A \times S \times Z \rightarrow \mathbb{R}$, will incorporate the new observations in the reward. The updated model from figure 3 is shown in figure 4.

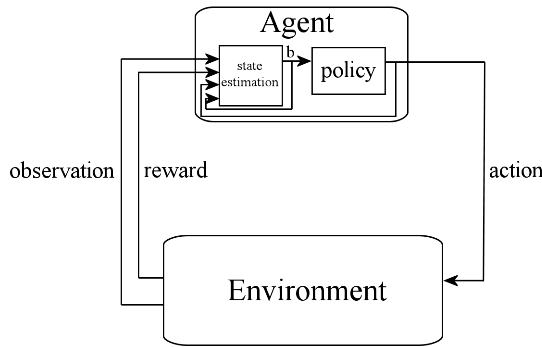


Figure 4: The POMDP model, since states are no longer accessible, they will have to be estimated from the observation, the reward and even the policy

Example. *Illustrating this with an example of a robot that has to move over a ledge in a straight line to reach its next way point. If the robot's decision maker is able to directly access the state it is in it would just execute the action for moving straight forward and gaining the reward that corresponds with getting in that state. Now consider the forward move of the robot is slightly off to the left, because of dirt on his left wheel or other factors the robot is not aware of or has any control of. If it executes the action of moving in a straight line, it would then fall off the edge. The observation function O will help the robot dealing with that error in his movement, by predicting that when the robot just moves in a straight line, it will have a higher probability of falling off the ledge instead of reaching its destination. Similar the reward R, which incorporates this possibility of falling off will predict a lower reward by executing the action of moving in a straight line. More actions are possible, each gaining the robot a different reward. Ideally the maximum reward would be predicted by moving a bit to the right, compensating the error, resulting in more or less a straight line. Since the decision maker will take the best action, it will choose that action and it is still able to reach its destination.*

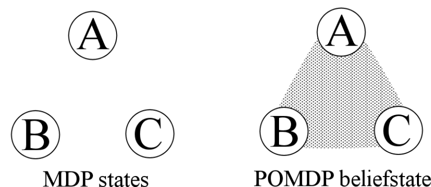


Figure 5: The MDP state space versus the POMDP state observations. In a COMDP the agent is either in state A, B or C and knows it. With a POMDP the agent can still be in A, B or C, but also in between, moreover the controller cannot know the exact state. The grey field represents the belief simplex, everwhere the agent can be in.

The expressiveness of the POMDP models however costs a lot of computational power. Where the decision maker in a COMDP model had to base its best action only on the state he was in to reach another state, in a POMDP it will have to deal with taking the best action given a probability distribution over the entire state space of reaching a combination of states described by another probability distribution. While the state space can be discrete, the probability distribution is continuous. Figure 5 illustrates this difference.

3.1 Beliefstates

Finding the optimal policy for a POMDP, is much more complicated than in a COMDP. Recall from section 2 that the states of a MDP are Markov and that therefore the optimal policies for a MDP are defined using the current state only instead of the entire history H. In a POMDP model, the decision maker does not have access to these Markov states. Thus, while not being able to use the Markov assumption, the entire history of observed states need to be mapped to the actions, $\pi : H \rightarrow A$, to find the optimal policy. Basing the optimal policy only on the last observation: $\pi : Z \rightarrow A$[6], or probabilistic policies over the last observation: $\pi : Z \rightarrow \Pi(A)$, can have practical solutions for some problems, but they can also be very poor.[3]

The optimal policy can be truly optimal when it remembers the entire history of the process, not just a part of it.[23, 1] However it is possible to compress the entire history to a summary state, a beliefstate, which is a sufficient statistic for the history. Using this statistic the history will be a probability distribution over the state space: $B = \Pi(S)$. The optimal policy can then be defined again over the entire history, $\pi : B \rightarrow A$. Unlike the real history however, the beliefstate B is of fixed size, while the history grows with each time step. With the beliefstate, the Markov property is regained for a POMDP. The notation for the beliefstate probability will be $b(s)$, the probabilities for each $s \in S$.

3.1.1 Bayes' rule

Bayes' rule can be used to get the update function for the belief states that are necessary for POMDPs [1]:

$$b_z^a = \frac{o(a, s', z) \sum_s \tau(s, a, s^{'}) b(s)}{\sum_{s,s''} o(a, s'', z) \tau(s, a, s'') b(s)} \tag{8}$$

This way the next beliefstate depends only on the previous beliefstate and the direct action taken to get into the next beliefstate. In other words it is a Markov state space.

3.2 Value Iteration for POMDPs

By describing the history of observations as a beliefstate, it is possible to convert the discreet space POMDP to a continue space COMDP, using the beliefstate as the states. With this the dynamic programming approach, value iteration can be used to evaluate the Values for taking an action given that the agent is in a beliefstate. The only thing that needs to be defined are the successor states, which were s' and are, in this case when using a beliefstate, B', a set of . Since there are only a finite amount of observations Z in a discreet POMDP the successor states B' can be defined as:

$$B'(b,a) = \{b_z^a | z \in Z\}$$

The beliefstate after doing a, when observing z.

Then the Value Iteration from section 2 is transformed to sum the expected rewards not only over all possible successor states, but also over the present state, weighted by their probabilities of the beliefstate. It also includes the observation fuction $o(a, s', z)$, providing a probability distribution of the likelihood that z is observed when the agent tries to go to successor state s'.

$$V_n^*(b) = \max_{a \in A} V_n^{*,a}(b) \tag{9}$$

$$V_n^{*,a}(b) = \sum_{s \in S} b(s) r(s,a) + \gamma \sum_{s \in S} \sum_{s' \in S} \sum_{z \in Z} b(s) \tau(s,a,s') o(a,s',z) V_{n-1}^*(b_z^a) \tag{10}$$

These values are calculated iteratively for a finite horizon $t = \{0, .., n\}$: Yielding the optimal policy:

$$\pi^* = \underset{a}{\operatorname{argmax}}\ V_n^{*,a}(b) \tag{11}$$

To calculate the value function for the finite horizon, the difficulty of the infinite continue state of the probability distribution over S has to be overcome. Sondik[11] has shown that the optimal value function for any finite horizon is Piece-Wise Linear and Convex(PWLC). This means that in order to calculate the optimal value function, only a set of linear functions, Ψ, will have to be calculated, which describe the value function. An example of such function for a two-state POMDP is shown in figure 6.

The optimal value function is then simply the maximum of all those linear functions, expressed as the bold line in the figure. Cassandra[1] shows that these lines can represent $\psi^{a,z}$, where they indicate the expected reward for doing an action a when observation z will be observed. A summation over all possible observations in Z for that action, will yield the real value function for a.

$$V_n^{*,a}(b) = \sum_{z \in Z} b \cdot \psi_n^{a,z}(b) \tag{12}$$

With equation 11 the agent will be able to choose the best action by just evaluating the maximum of two PWLC functions for the given beliefstate the agent is in.

For a relatively low number of individual states that are in the set S, this method will solve the POMDP exact, and is relatively doable. There are a couple of methods of computing and pruning Ψ more efficient. However,

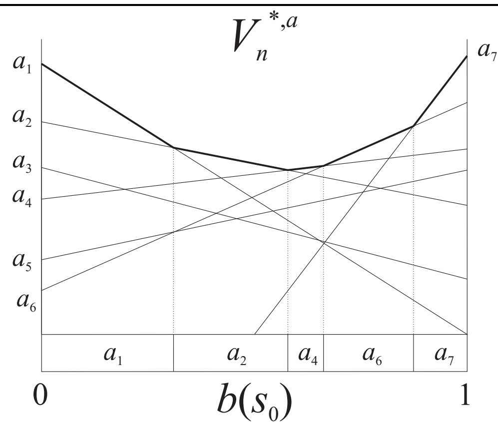


Figure 6: Example of calculated reward lines for doing different actions $\{a_0..a_7\}$ by using equation 12, together they form a Piece-wise Linear and convex function. This example is for a two state beliefstate $b(s_0)$ and $b(s_1)$, only $b(s_0)$ is shown. Since $b(s_0)$ denotes the probability of being in state s_0, at the right side the agent will be for 100% sure its in s_0, at the left side, where $b(s_0) = 0$ it will be 100% sure its in s_1. Instead of having to calculate the value function for every possible beliefpoint in the continue interval $[s_1, s_0]$, it can be expressed by these seven lines. In the block at the bottom of the graph are the intervals shown which a_i will yield the best reward.

each time a state is added to S a whole dimension is added to the continue beliefstate space. For large state spaces these methods can become too complex to be calculated.

For approximating the value function, Reinforcement Learning can be used. Methods that implement this can solve POMDP problems better as they can also solve POMDPs that have larger state spaces. Its also a solution for non-deterministic POMDPs, like simulations where the model is sampled while running it. Some examples are Lin-Q and k-PWLC.

3.3 Heuristic Approximation for POMDPs

Solving the POMDP with an exact Value function algorithm is not very doable due to the large state space, problems typically have. Therefore approximations for calculating the optimal value function can be implemented. Hauskrecht [2] reviewed many heuristic value function approximators and pointed out their performance. Although trying to bind the approximate value function as close to the PWLC exact bound seems a good way to evaluate an approximation, in the end what matters most is the control performance of the agent, regardless its value function. It seemed from Hauskrechts work that these two things are not directly related. On the contrary, the distance from the bound was not the most important thing, it was the shape. Q-MDP and the fast informed bound controllers seemed to work best on his simulation, while they were not as close to the optimal value function as the other approximations, the reason for the good performance though is their shape of the value function. With Q-MDP and the fast informed bound controllers the shape of the value function is also PWLC.

3.3.1 Most Likely State

When using the beliefstate as the state representation, since its a sufficient statistic for the history it is the best method of describing the current state. A way to solve this might be that the state that has the most probability mass of the beliefstate, would be the actual state the agent is in.

$$s_{MLS} = \underset{s}{\operatorname{argmax}}\ b(s) \tag{13}$$

Randomly picking one state s to be s_{MLS} if there are more that have the same maximum probability mass. With this s_{MLS}, the optimal policy would then be:

$$\pi_{MLS} = \underset{a}{\operatorname{argmax}}\ Q^{MDP}(s_{MLS}, a) \tag{14}$$

By using the Q_{COMDP}, the agent assumes to be in state s_{MLS}, also for the future rewards, since they no longer depend on the POMDP parameters

O and Z. However after the agent executes his action a, the beliefstate will be updated and a next MLS will be taken from it. It will not be necessary that $\tau(s_{MLS}, a, s'_{MLS}) > 0$, since agent will really move from one beliefstate to the next. Discarding the chance an agent will be in every other state but the state with the maximum probability mass, will reduce the computation complexity significantly, and can still be fairly optimal if the assumption is made that agent will probably be in one state at a time anyway.

3.3.2 Q-MDP

With Q-MDP not only the Most Likely State is taken into account when finding the best action. After calculating the Value-action pairs some way the probability distribution over the state space, which are stored in the beliefstate, are multiplied for against every $Q(s, a)$ for each state. Then summed together it gives a weighted average for an action of the values for that state and the probability for that agent to be in that state. This weighted average is generally much better since it takes all probabilities of the beliefstate into account, not only the maximum one. Then the maximum action of all of those weighted value-action values is taken being the optimal policy.

$$\pi^{QMDP} = \underset{a}{\operatorname{argmax}} \sum_{s} b(s) Q^{MDP}(s, a) \tag{15}$$

It is shown by Littman et al. [6] that the Q-MDP value function is also PWLC.

4 Green Light District Simulator

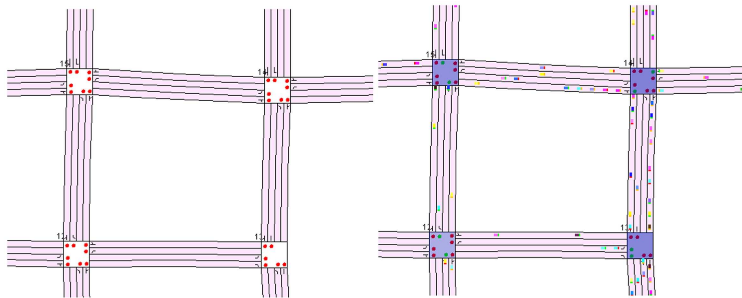


Figure 7: Green Light District Simulator, GLD, this is where all the road users move endlessly. The left picture shows a piece of Jillesville prior to the start of a simulation run, the right picture of the same piece but during a run.

The simulator used is a freely available traffic simulator, the Green Light District (GLD) traffic simulator.[21, 22] The main entities in the simulations are the road users moving through a traffic network and traffic light controllers controlling individual traffic lights, denoted by *tl*, at traffic light *junctions*. The road users move on *roads* consisting of different *drive lanes* going in different *directions* at a *junction*. Every *drive lane* consists of a discrete number of positions, denoted by *pos*, each of which can be occupied by a road user. As in the real world, particular drive lanes and corresponding traffic lights constrain the direction in which a road user can go. For example, the left lane may be for road users that turn left, the right lane may be for road users that go straight or turn right see figure 8 for a visual representation of an example *junction*.

4.1 State Representation

For the Traffic Light domain there are at least two types of state representation, as described by Wiering et al.[21, 22]:

- Road user-based state representation, this represents the world state from the perspective of individual cars. The value functions for individual cars and the traffic light decision is made by combining the value functions of all cars around the junction. Thus, this is a multi-agent approach. The cars do not have to represent the value functions themselves though, moreover the cars don't generally know their value functions [21, 22].

- Traffic light-based state representation in which you have to represent all possible traffic configurations around a traffic light junction. This means that every traffic light junction has to learn value functions that maps all possible traffic configurations to total waiting times of all cars. This will lead to an enormous state space and is therefore not advisable [18].

With regard to the working with partial observability, which also requires a big increase in belief state space, it was desired to use as small a state space as possible to begin with. The Car-based approach was therefore the best.

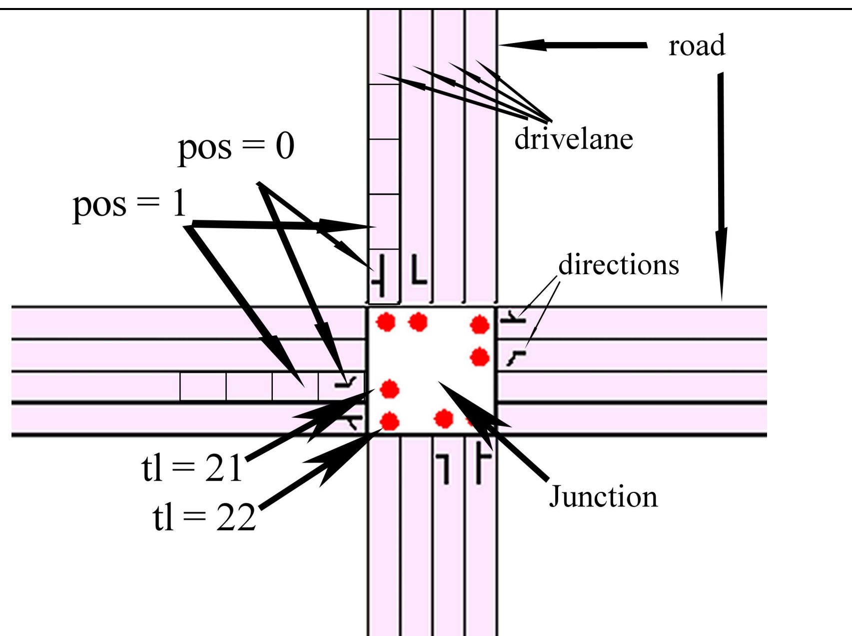


Figure 8: An example of an average junction in the simulator with four roads attached to it. Each road has four drive lanes. The two that go to the junction has a traffic light associated to it. The directions denote to which road the road user can cross the junction. The positions on the individual drive lanes range from $pos = n$ at the end of an n blocks drive lane and $pos = 0$ at the beginning. The combination of the position and the unique tl identifier forms the state s.

The traffic light controller or *tlc* will maintain a state representation for each car approaching a traffic light *tl*, at a position *pos* in front of the traffic light. A *pos* of 0 means that the road user is at the traffic light. Each state s can be described as a tuple:

$$s =< pos, tl > \tag{16}$$

where s and tl are unique identifiers to the infrastructure, while *pos* is unique to the *drive lane* connecting to the tl. There is always room for extension in how you represent the state s, for example an addition to incorporate the congestion state bit, *csb*. [15]

Although the road user based-approach intuitively suggest that the actions taken are about road user movement, it is actually still about traffic light control. The controller does not have control over the road users themselves, but can assume that the road users will move if able to move. The actions themselves are about having a green or red light. Therefore we define the set of actions a to be:

$$a \in \{red, green\} \tag{17}$$

5 MDP Implementations of Reinforcement Learning for Traffic Light Control

As stated in chapter 2 Reinforcement Learning usually formalizes the learning problem as a Markov Decision Process (MDP). Most R_t algorithms (such as Q-learning[13, 8]) define and learn a value function $Q(s, a) : S \times A \rightarrow \mathbb{R}$, whose values directly represent the direct reward and the discounted future reward which can be gained in state s, by executing action a. Once the optimal value function Q^* has been found, the optimal policy for the agent simply corresponds to: $\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s, a)$.

5.1 Standard

Model based reinforcement learning uses the different types of actions to learn the action-value function given a specific state. These values are defined by the utility(reward) it gets from taking an action in a certain state and performing a certain transition to another state.

$$Q(s, a) \leftarrow \sum_{s'} \tau(s, a, s')(r(s, a, s') + \gamma V(s')) \tag{18}$$

where $V(s) = \sum_a P(a|s)Q(s, a)$.

A specific transition model and reward function is used in order to be able to control the traffic lights with Reinforcement Learning.

The state transition probabilities are calculated from the number of occurrences of road user transitions divided by the number of occurrences of the initial states leading to the transition. This is a maximum likelihood model. The state transitions themselves will not be provided up front, but will be sampled along the way.

$$\tau(s, a, s') = \frac{count(s, a, s')}{count(s, a)} \tag{19}$$

The "support" is used to have a measure for the quality of the found state transition probabilities for a specific state. This is not part of the standard RL/TLC vocabulary, but will be useful later on. The support ϕ is the number of times the state has been visited and denoted by:

$$\phi(s) = \sum_{a} count(s, a) \tag{20}$$

If a road user stays at it's location without moving, that controller will receive a penalty of -1. Whereas if the road user can move the controller will receive a reward of 0. This simulates the intuition of waiting longer being bad, since more cars waiting gives a higher penalty to that traffic light configuration. The reward function can be defined as:

$$r(s,a,s') = \begin{cases} -1 & \text{if } s = s' \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

It is less likely that a red light is chosen in that situation from the different set of configurations A_n for traffic light n, note that there are no possible configurations that would present unsafe traffic light configurations. The optimal policy is one of those configurations:

$$A_n^{opt} = \max_{A_n} \sum_i \sum_{s \in L} (Q(s, green) - Q(s, red)) \tag{22}$$

where L is the set of all cars currently in front of traffic light i at junction n. A_n^{opt} is the optimal traffic light configuration at junction n.

5.2 Congestion

In previous work the problem of congestion [15] was dealt with. This was done in two specific ways, the most learning based method was expanding the state s to a tuple $s =< pos, tl, csb >$. Here the csb is the bit that indicates for a car if the next lane is congested or not. This is done by calculating a real-valued congestion factor c:

$$c = \frac{w_j}{D_j} \tag{23}$$

where w_j is the number of cars on the car's destination lane j, and D_j is the number of available positions on that destination lane j. csb is computed according to

$$csb = \begin{cases} 1 & \text{c} > \theta \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

where θ is a parameter acting as a threshold.
The other way that was implemented to tackle the congestion problem was by use of an heuristic approach. Rather than using the real-valued c to obtain an additional state bit, it is used in the computation of the estimated optimal traffic light configuration:

$$A_n^{opt} = \max_{A_n} \sum_i \sum_{s \in L} (1 - c(s))Q(s, green) - Q(s, red)) \tag{25}$$

where $c(s)$ is the appropriate congestion factor for each car in L. In other words, the congestion factor c is subtracted from 1 such that the calculated gain for an individual car will be taken fully into account when its next lane is empty (it is then multiplied by 1), or will not be taken into account at all if the next lane is fully congested (it is then multiplied by 0).

Improving the state space by adding elements have a positive influence on the optimal decision, as was the case with the developed congestion state

bit. However the cost of creating larger state spaces have as a downside the increase in complexity. Apart from an increase in computation speed with a more complex value function there is also the speed of convergence. With more complex state spaces it can take longer for the controller to be able to make a well founded choice of action. A trade-off between speed and optimality has to be made. Keeping in mind the Partial observability the most simple state representation will be used. For more information about state representations in this domain see [15] and [21, 22].

6 POMDP Implementations of Reinforcement Learning for Traffic Light Control

Up till now the GLD simulator was fully observable with respect to state access. To be able to test a POMDP on this simulator, the state information has to be made partially accessible through observations. This can be done in two ways. One is limiting the amount of sensors, so that some road blocks on each drive lane are completely unobservable. The other is by adding random noise to sensors connected to the blocks.

6.1 Sensor noise

Real world sensors are not always 100% accurate. This is because the object that is being observed by a sensor differ from one another. In traffic for example the size and weight of a vehicle can influence detection. There are many types of sensors in the traffic environment. Nihan et.al. [9] showed in their report that dual-loop detectors, and in less extend the more common single-loop detectors, are very accurate in detecting individual road users. However when other features of the road users are taken into account, like for example the speed, performance is less than optimal.

Misinterpretations by sensors due to the complexity and diversity of the objects can be generalized to sensor noise. Sensor noise can be modelled as white noise, removing at an independent random interval a sample from the sensor information which let the system believe the sensor did not measure an object. Or this noise can be dependent on events that take place, for example after a truck just passed by, the chance of detecting a normal car is less than normal.

If the traffic controllers would be implemented in a real world environment, being able to handle these noise models is desired. Although most of the times sensors work all right.

6.2 Incomplete sensor information

Another difficulty arises when using sensors on a lane. The sensors cannot be placed too close to each other, let alone next to each other, or by doing so the effectiveness of each individual sensor will decrease. How can one know if two sensors that are next to each other are measuring one truck or two cars? Even when using cameras this problem cannot be solved like this. Figure 9 shows the differences between a fully observable road, where S are the actual states and a partial observable road, where there are not many sensors and the the sensors that exist are susceptible to some noise e.

The way the problem is dealt with in this thesis is by means of interpolating the whereabouts of a road user between two sensors by means of chance. At the start of the lane, there will be a double loop sensor, detect-

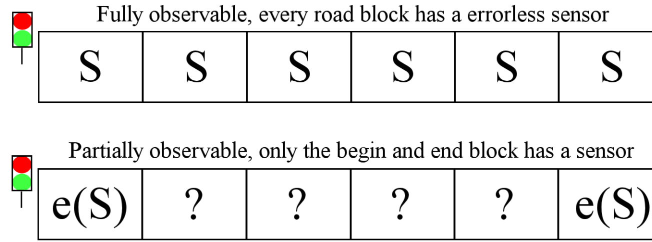


Figure 9: Sensors on fully and partially observable roads.

ing road users and detecting its speed by calculating the difference in time between the road user crosses the first loop and the second loop. Then with some model the road users speed will be extrapolated over the follow-up time steps. Assuming the speed is estimated, the location can be calculated. After the road user arrives at the traffic light, there will be another sensor, detecting the road user, correcting the interpolation of its speed with respect to the other road users on the lane to improve the quality of the interpolation.

6.3 State Estimation

The exact state cannot be known, thus there is incomplete knowledge of the road users speed and position. As stated before the belief state is a mean to keep a Partial Observable problem a Markov Decision Process. It is needed that the beliefstate is a sufficient statistic of the history.

6.3.1 Implementation

It is now necessary to create a beliefstate for GLD roads. In order to get a good beliefstate of a drive lane b_{dl} it was first necessary to create a beliefstate for the individual road users b_{ru}. To create b_{ru}, sensor information can be used in combination with assumptions that can be made for a road user.

The actions u are the actions of the road user and the sensor information can be seen in figure 9. There are sensors on the beginning and the end of the road. Given that there is knowledge of a road user being on the beginning of the road on time step $t = 0$ then the assumption can be made, given for example that the different u the action of moving 1, 2 or 3 steps forward, thus implicitly denoting the speed of the road user. This means that the road user is on either position 1, position 2 or position 3 of the road on time step $t = 1$. With the same logic the whole b_{ru} of all cars can be done for all time steps.

$$b_{ru}(s) = p(s_t|Z_t, U_t) = p(z_t|s_t) \sum_{s_{t-1}} p(s_t|u_t, s_{t-1}) p(s_{t-1}|Z_{t-1}, U_{t-1}) \quad (26)$$

Particle filters are used in the continue state space domain counterpart [19].

Since the GLD MDP problem is a multi-agent problem, calculating only belief states of the individual road users, with an assumption that those individual road users are not dependent on each other position is not correct. Therefore b_{dl} will need to be constructed, where the positions of each road user does depend on that of the others on the lane.

To create the b_{dl}, all the b_{ru} for a specific drive lane must be combined. Three assumptions have to be made in advance. Firstly road users that spawn on a later t cannot pass road users that spawned on an earlier t. The second assumption is that road users cannot be on top of each other. The third, most important, assumption is that the road users have to be on the road. Road users spawned on an earlier time step must leave room behind them so that all spawned road users on a later time step have a spot on the road. The way b_{dl} was implemented, to incorporate these three assumptions, can be seen in algorithm 1.

6.4 Decision Making

As was discussed in section 3 there are several ways of solving a POMDP. All of the methods use the beliefstate from the previous sections as state representation. In related literature two curses for POMDPs are mentioned that make an exact solution impractical to compute, these are The Curse of Dimensionality [4] and The Curse of History [10]. An optimal policy is computed over an n-1 beliefstate space, where n is the amount of states. An infrastructure in GLD, like Jillesville [21, 22, 15] can have roughly about 5000 states. A simple pruning of relevant states can be made by the assumption that road users cannot switch lanes, or when its still desired for them to switch to lanes directly next to them and going into the same direction, that they cannot switch to lanes that are not directly connected. This reduces the amount of relative states to only those that are on the lane, and perhaps next to the lane, which are for a typical lane, between the 15 and 30. Still the optimal policy will have to be calculated over a 14 to 29 dimensional beliefstate space for each lane in the infrastructure. when calculating the optimal policy you can approximate either the value functions, the beliefstate or both.

In section 3, possible candidates for implementation have been discussed. Since large state spaces are an issue, the methods chosen are Q-MDP and MLS. Q-MDP gives a good performance because of its PWLC property of the value function and is still relatively easy to compute. MLS is a more simple heuristic method assuming that the most likely belief point in the

Algorithm 1 Calculate b_{dl} from all b_{ru} of the road users on a drive lane.

1: set q_0 to be a roaduser state tuple $< r, s >$ with $s.pos = -1$ and $Pr(q_0) = 1$
2: Q is new set existing of $\{q_0\}$.
3: R is the set of road users on the drivelane.
4: **for all** $r \in R$ **do**
5:   Q' is new empty set.
6:   set the boolean $found(r)$ to false.
7:   set the last possible position for this road user to be $LastPossiblePos = |R| - num(r) \cdot 2 - 1$.
8:   calculate b_{ru} for r
9:   **for all** $q \in Q$ **do**
10:     lastpos is the position of the state of the last element added to q.
11:     **for** $j : lastpos \rightarrow LastPossiblePos$ **do**
12:       copy q to q'
13:       $Pr(q') = Pr(q)$
14:       **if** position j equals LastPossiblePos **then**
15:         add $< r, s >$ to q' with $s.pos = j$
16:         **if** $found(r)$ is false **then**
17:           $Pr(q') = Pr(q') \cdot 1$
18:         **else**
19:           $Pr(q') = Pr(q') \cdot b_{ru}(E); E = \{S \cap s.pos \geq j\}$.
20:         **end if**
21:       **else if** $b_{ru}(s.pos = j) > 0$ and the position j is not used in q' **then**
22:         add $< r, s >$ to q'
23:         $Pr(q') = Pr(q') \cdot b_{ru}(s)$
24:         set $found(r)$ to true
25:       **else if** $b_{ru}(E) > 0; E = \{S \cap s.pos < j\}$ and the position j is not used in q' **then**
26:         add $< r, s >$ to q'
27:         $Pr(q') = Pr(q') \cdot b_{ru}(E); E = \{S \cap s.pos < j\}$
28:         set $found(r)$ to true
29:       **end if**
30:     **end for**
31:     add q' to Q'
32:   **end for**
33:   set $Q \leftarrow Q'$
34: **end for**
35: **return** $b_{dl}(q) = Pr(q) : \forall q \in Q$

beliefstate is good enough for calculating the optimal policy. There are many other approximations for a discreet environment [2]. For example point-based algorithms [12], however most of these algorithms rely on doing information gathering actions, which cannot be performed by the traffic light controllers, or are still very computational intensive.

6.4.1 All in Front implementation

To provide a baseline method, a very naive and simple method is needed for comparison. The All in Front, or AIF, method is exactly that. It assumes that all the road users that are detected by the first sensor are immediately at the end of the lane, in front of the traffic light. The road users are not placed on top of each other. Then the decision is based on this simplistic assumption for the configuration of the traffic lights.

6.4.2 Most Likely State implementations

In this section two kinds of Most Likely State implementations are discussed. The first is the traditional MLS, the one described by equations 13 and 14. Using the individual road user beliefstate for this MLS equation 13 will become:

$$s^{MLS} = \underset{s}{\operatorname{argmax}}\ b_{ru}(s) \tag{27}$$

and

$$A_n^{MLS} = \max_{A_n} \sum_i \sum_{j \in L} (Q(s_j^{MLS}, a = green) - Q(s_j^{MLS}, a = red)) \tag{28}$$

for each junction n, traffic light i, and all road users j on drive lane L. This will be the most naive method implemented, using only the individual road user belief states. There was also a beliefstate defined for a drive lane. Summarizing not only the probabilities of the individual road users but also their joint probabilities, eliminating not possible configurations of road users on a lane with the two assumptions that all road users will have a spot on the drive lane and that the order of the cars stay the same, they cannot jump over each other. This leads to a much better beliefstate, the one over the road users queueing for a traffic light. With this beliefstate the state can be seen as the possible queues of road users for that drive lane. The MLS approximation for this queue is called the Most Likely Queue in the thesis, or MLQ, the MLS for this multi-agent problem.

$$q^{MLQ} = \underset{q}{\operatorname{argmax}}\ b_{dl}(q) \tag{29}$$

and

$$A_n^{MLQ} = \max_{A_n} \sum_i \sum_{s_j^{MLQ} \in q^{MLQ}} (Q(s_j^{MLQ}, a = green) - Q(s_j^{MLQ}, a = red)) \tag{30}$$

where n, and i are the same as in 28, but the states $s \in q$ are the individual states of road users, derived from the set of $q^{MLQ} \subset S$, that together have the maximum likelihood compared to other configurations q. The indices for s in $s_1, s_2, .., s_n$ = q directly correspond to the indices for the road users $ru_1, ru_2, .., ru_n$ that are on drive lane L.

6.4.3 Q-MDP implementation

Recall from section 3, equation 15, that the optimal QMDP policy is taken over the best action from $Q^{MDP}(s, a) \cdot b(s)$. An implementation will offer two options regarding the belief states. The b_{ru} or the b_{dl} belief states can be used. When using b_{ru} the states can directly be used since the individual road user beliefstate is defined as a probability distribution over S: $B_{ru} \rightarrow \Pi(S)$. However for a more optimal solution, b_{dl} would be a better choice. Since the drive lane beliefstate is defined as a probability distribution over possible queue configurations of the states on a specific drive lane: $B_{dl} \rightarrow \Pi(q); q \subseteq S_{dl} \subset S$, the probabilities for each individual state have to be calculated from all queues $q \subseteq S_{dl}$.

Another Beliefstate $b_{dl,ru}(s)$ can then be defined as a probability distribution over the state space given road user ru is on state s of that drive lane, given the combined drive lane belief states.

This belief state, which combines all individual road user belief states together, can then be used in the optimal traffic light configuration policy, using Q-MDP $A_n^{QMDP,opt}$ for a junction n:

$$A_n^{Q-MDP,opt} = \max_{A_n} \sum_i \sum_{j \in L} \sum_s b_{dl,j}(s)(Q(s, green) - Q(s, red)) \tag{31}$$

for all traffic lights i on junction n and all road users j in L.

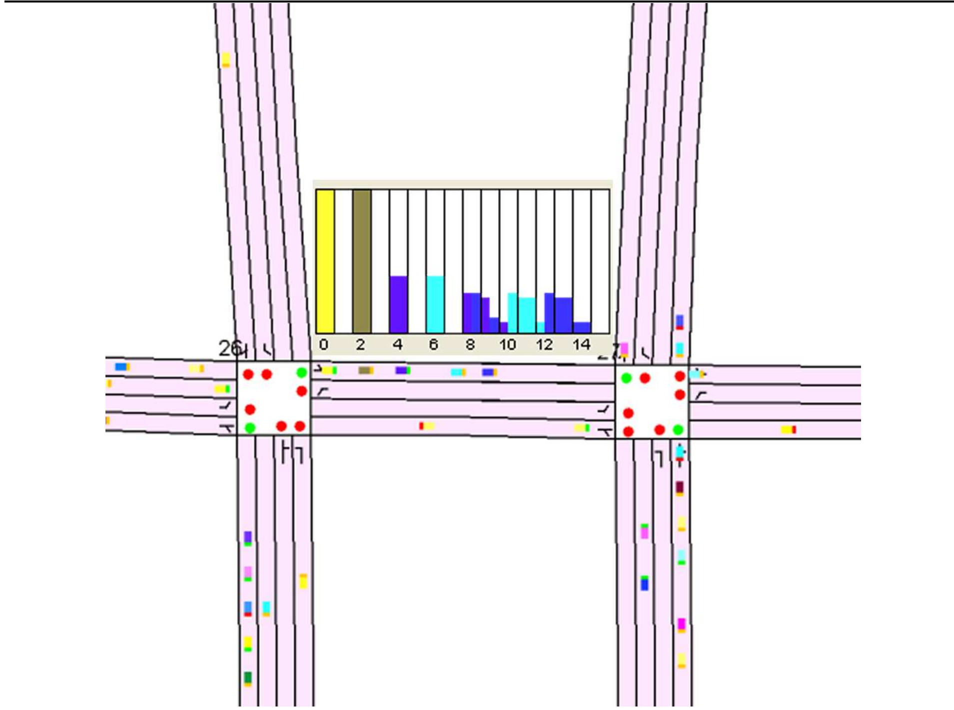


Figure 10: The probability distribution of b_{dl} is shown for a drive lane of the Jillesville infrastructure. The gains that are stored in $Q(s, green)$ are multiplied with the values of b_{dl} that can be seen in the figure.

6.4.4 Beliefstate Approximation

Implementing the MLS, MLQ and Q-MDP methods gave rough empirical results on belief states sizes, the number of different probability distributions over the state space S could be over more than 170 thousand for the Q-MDP method, which used all of them to calculate the optimal policy. Huge slowdowns of the simulator were the result or even the inability to run the simulator due to the lack of memory required. There was the need of a method of reducing the number of different beliefstate probabilities.

Because the algorithm that calculated the combined belief states of all road users was effectively calculating the permutations of the positions of the road users on the drive lane, many of those resulting belief states had a very low probability of actually occurring. A logical approach was to just prune the most unlikely configurations of road users, to make the set of all belief states smaller, removing belief states with the lowest probability mass until a certain threshold was reached. The extreme way would have been to prune it all but to one state, in which the resulting belief state would have been the MLQ beliefstate. The other side was to only delete the

probabilities up to 0.05 of the total mass. Setting a good threshold however is very difficult, since its a trade-off between performance and hypothetical better results. Algorithm 2 was used to delete the lowest probabilities. One way to define θ for this environment was to instead of making a fixed one, making one that was related on the number of road users and the size of the drive lane. Intuitively less different belief states are possible when the lane is more filled with road users because the degrees of freedom decreases. Results of the beliefstate reducing algorithm are shown in table 1 and 2. Notice from the two tables that the lower bounds of the number of states are much lower on the LongestRoad infrastructure while the upper bounds are much higher, the number of possible combinations on the longer drive lanes can be exponential in the number of road users when the road users are far enough from each other so that not too many combined configurations are cancelled out.

method	average	max
MLQ	121	156
QMDP with $\theta = \frac{\lvert ru \rvert}{\lvert blocks \rvert}$	2347	4698
QMDP with $\theta = 0.05$	20453	52069

Table 1: Number of different beliefstate probabilities for different methods and θ on the Jillesville infrastructure

method	average	max
MLQ	13	39
QMDP with $\theta = \frac{\lvert ru \rvert}{\lvert blocks \rvert}$	111	10311
QMDP with $\theta = 0.05$	2005	173723

Table 2: Number of different beliefstate probabilities for different methods and θ on the LongestRoad infrastructure

Algorithm 2 Remove the lowest probabilities from a beliefstate up to θ.

1: $skipped = 0$
2: **while** $skipped < \theta$ **do**
3: $\quad d = \operatorname{argmin}_s b(s)$
4: $\quad$ add probability mass $b(d)$ to $skipped$.
5: $\quad$ remove $b(d)$
6: **end while**
7: normalise b.

6.5 Learning the Model

Now there are solutions to solve the POMDP, given that the model could still be learnt in a completely observable way as is done in the COMDP. Yet for a possible realistic application, this assumption cannot hold. The model cannot be learnt by counting system states transitions, which cannot directly be accessed by a POMDP controller. Instead all the transitions between the belief state probabilities of the current state and the probabilities of the next belief states should be counted to be able to learn the model. Recall however that belief states are continue over the state space so there will be an infinite number of state transitions to count each time step. This is not possible. An approximation will therefore be needed.

This approximation would be to learn the model without using the full beliefstate space, but just using the beliefstate point that has the highest probability mass. Using the combined road user beliefstate b_{dl} is preferred, the most likely states in this beliefstate are s_{MLQ}. Assuming the most likely state is the real state, the controller could simply count the transitions of road users as if the state was fully observable by moving the road users one step ahead in time from their state s_{MLQ} and counting them as in 19:

$$\widehat{\tau}(s,a,s') = \frac{count(s_{MLQ},a,s'_{MLQ})}{count(s_{MLQ},a)} \tag{32}$$

Though this approximation seems less than optimal, it provides a way to sample the model and the intuition is that after many samples it will ultimately be that $\widehat{\tau} \cong \tau$.

7 Test Environment and Statistics

As seen in section 4 the GLD-simulator was used in order to test and compare algorithms. In this section there will be an overview of the traffic networks used and the different statistics used in order to compare the algorithms.

7.1 Test Domains

The algorithms made were tested on three different traffic networks, namely the Jillesville traffic network, the LongestRoad and the Simple traffic network with only one junction.

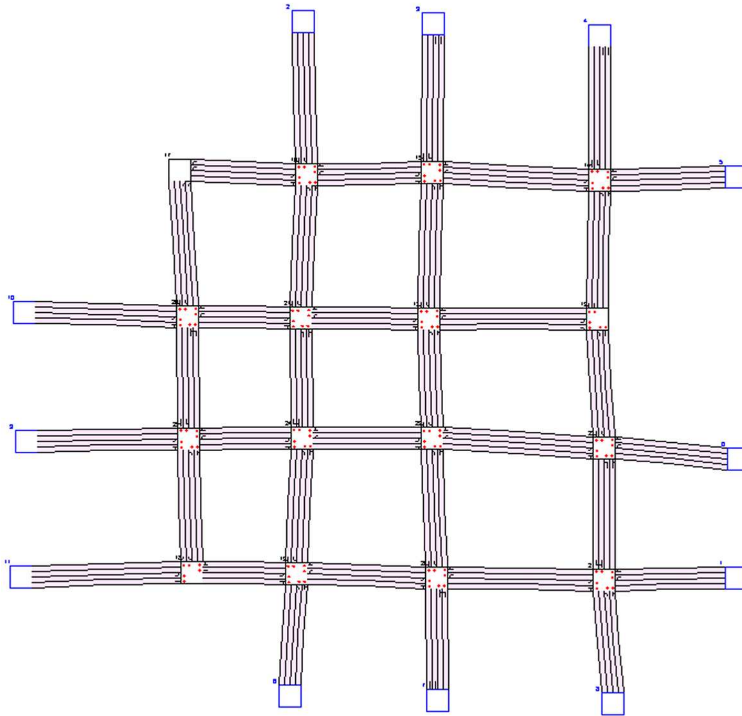


Figure 11: The Jillesville infrastructure.

Jillesville, see figure 11, is the same network as used by Wiering [21, 22] and [15]. Since it was used in previous work it is interesting to use it for partial observability to give a comparison between COMDP and POMDP performance. Jillesville is a traffic network with 16 junctions, 12 edge nodes or spawn nodes and 36 roads with 4 drive lanes each.

LongestRoad, figure 12, is a much more simple traffic network than Jillesville because it only has 1 junction, 4 edge nodes and 4 roads with 4 drive lanes each. The biggest change though in this set-up is that the drive lanes themselves are a lot longer than in Jillesville making the chance

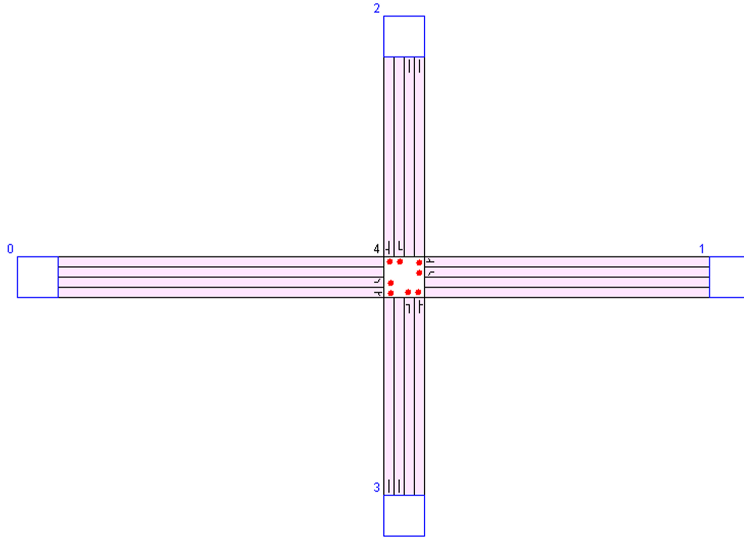


Figure 12: The LongestRoad infrastructure.

to be wrong with the beliefstate of the road user a lot higher. This was used to test the robustness of the POMDP algorithms.

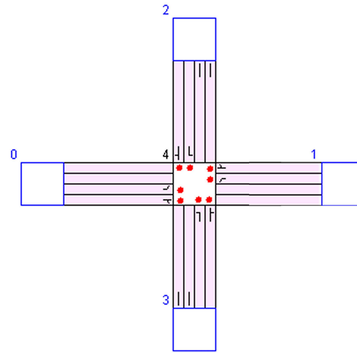


Figure 13: The Simple infrastructure.

The Simple Traffic network, figure 13, was developed after the LongestRoad one. It is identical to LongestRoad as it also has 1 junction, 4 edge nodes, 4 road with 4 drive lanes each. But the change was made to the length of the drive lanes, there is now only room for 6 cars. This has the effect that this traffic network is the simplest network. Making the influence of partial observability on the decision making of the network the least. This is a good baseline for the POMDPs, since they should almost have the same information as the COMDPs and should therefore perform the same. More about the specific tests later.

7.2 Statistics

In previous work with Jillesville [21, 22, 15] the majority of tests was done based on the average traffic waiting time (ATWT). This showed how long it would take on average for a road user to get to the goal from their original spawn node. This was not the best performance statistic as was found out later. It has a couple of drawbacks including the fact that if a road user does not arrive at his destination and is standing in front of a traffic light forever, he will not be counted in the ATWT statistic. Thus many tests that came to a deadlock, where the traffic network and algorithm simply could not handle the amount of traffic that was entering the simulated city, performed better than when the tests did not come to a deadlock. In order to compare the different algorithms there is still need for some statistical tools though. So for this domain there are different types of statistical tools, namely, Average Junction Waiting Time and Total Arrived Road Users.

7.2.1 Average Junction Waiting Time

The average junction waiting time (AJWT) shows the time a road user has to wait on average at the junction. The longer it has to wait, the worse the result is. It reflects how long a road user has to wait in front of red lights in order to get from the spawn node to the goal node. This is similar to Average Traffic Waiting time, except that it has a smaller version of the drawback of ATWT. Still when the infrastructure is completely full it still gives some random results for waiting times, due to the fact that road users that never cross the junction are not counted.

7.2.2 Total Arrived Road Users

Total Arrived Road Users (TAR) is, as the name implies, the number of total road users that actually reached their goal node. In case of a total stand still of all road users this can reflect nicely if road users actually arrive at their goal and don't just stand still in front of a red light forever. For practical use it can still be used if runs in a series of runs come to a deadlock of the network.

7.3 Different Set-ups

The standard traffic network Jillesville was used for most of the testing and comparing of algorithms. The LongestRoad was made to have a better overview of how the POMDPs performed in a more difficult and higher partial observable environment. The simple traffic network on the other hand was used to verify a theory that was made after seeing the results of the Jillesville network. This traffic domain can be used to do simple checks on state spaces. To get different set-ups there is now an option of 7

different algorithms to compare. To have a small recapitulation, as described in detail in section 5, COMDP. As described in section 6, there were AIF, MLS, MLQ, QMDP, MLQ-MLQlearnt and QMDP-MLQlearnt. The tests consisted of one series of runs, with 10 runs per series and where the spawn rate of road users for the edge nodes was increased each subsequent series from 0.25 to 0.5 road users each cycle. All runs assume that road users change their speed uniformly which means that when a road user has a speed of 2 in the previous time step it has 1/3rd probability of changing it to 1 or 3 or keeping the same speed. When the road user has a speed of 1 or 3 in the previous time step, it has 1/2 chance to change its speed to 2 or stay the same speed. Another possibility is to use a Gaussian assumption of changing speeds; with the the parameter μ centered on the speed it had the previous time step and with $\sigma = 0.5$. This assumes that road users are likely to stay at their current speeds. Note that the actual movement is neither one of the two. In the next section there will be a discussion of the tests and their results in detail.

8 Experiments & Results

An overview of what will be discussed in this section is presented in table 3. There is a total of seven tests performed with a difference in the infrastructure it has been tested upon. Some are tested on multiple infrastructures, whereas others are run on only one.

Test	Different Algoritms used	Infrastructure
experiment1a	COMDP, AIF, MLS,	Jillesville
experiment1b	MLQ, QMDP	LongestRoad
experiment2a	COMDP,	Jillesville
experiment2b	MLQ-MLQlearnt, QMDP-MLQlearnt	LongestRoad
experiment3	QMDP+sensor errors	LongestRoad
experiment4	MLQ+sensor errors	LongestRoad
experiment5	COMDP, MLQ, QMDP Gauss vs. Uniform	Jillesville
experiment6	COMDP, MLQ, QMDP	Simple
experiment7	COMDP, SCOMDP, MLQ, QMDP,	Jillesville
	MLQ-MLQlearnt, QMDP-MLQlearnt	LongestRoad

Table 3: The different test configurations and comparisons done

8.1 Experiment 1: COMDP vs. POMDP algorithms

The results in figure 14 show that QMDP performs outstanding, even better than COMDP, with MLQ slightly behind the two top algorithms. AIF and MLS perform terribly as was expected due to their not so advanced heuristics. The highest throughput is thus reached by QMDP since that algorithm has the most Total Arrived Road Users. This test also shows that the robustness of QMDP is better since the maximum spawn rates are higher for QMDP before the simulation gets into a deadlock. Generally the algorithms MLQ, QMDP and COMDP perform almost as well, which is a good thing, because MLQ and QMDP are already solutions to the POMDP problem, even though the model is still learnt as though it was still completely observable.

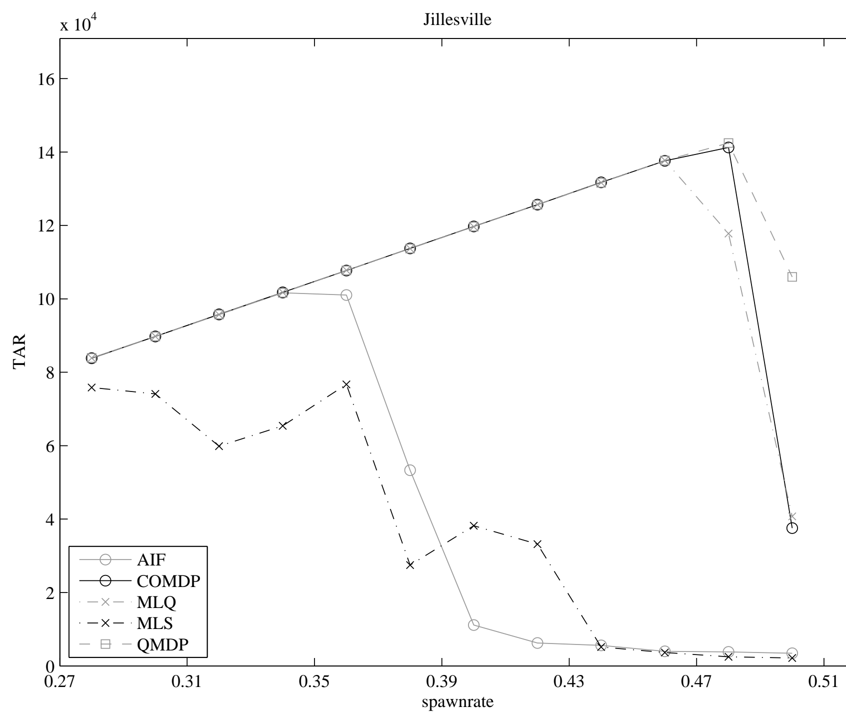


Figure 14: Result of Experiment 1a

When the roads are longer a slightly different result can be observed, see figure 15. In the Jillesville traffic network MLS was under performing in comparison with the other methods. On the LongestRoad network on the other hand the performance is not bad. Yet the COMDP is doing worse than all the POMDP methods except for AIF. What could be causing this bad behaviour of COMDP compared to the algorithms that function with partial observability? In theory COMDP should be the upper bound, since its state space is directly accessible and the Reinforcement Learning approach should then perform nearly optimal. One suggestion would be that the speed of convergence of the COMDP algorithm is worse on this infrastructure compared to the other infrastructures because of the greater length of the roads. Longer roads provide more states and since a road user will generally not come across all states when it passes through, more road users will be needed in order to get enough counts for each state, see equation 19. The POMDP algorithms have less degrees of freedom when observing movement of road users with respect to the real movement of the road users. The transition probabilities that are derived by counting in this situation will get a higher support ϕ faster. The support ϕ was defined in section 5.1, equation 20

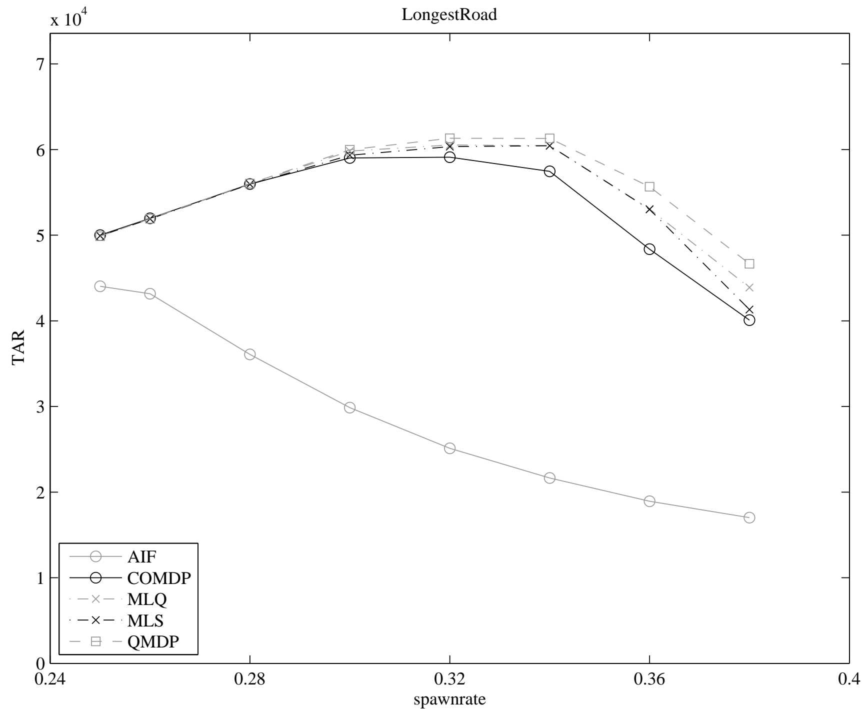


Figure 15: Result of Experiment 1b

8.2 Experiment 2: Learning the model with MLQ vs. COMDP

Experiment 1 has shown a good result in making decisions with partial observability. It is interesting to see what results learning the model with MLQ states, instead of the real states, will give. To test this the MLQ and QMDP methods now learn the model with MLQ, whereas COMDP still learns the modelled with the real state information.

The results on Jillesville are shown in figure 16. It can be seen that the learning with MLQ works extremely well displayed by comparing the highest points in the figure and seeing that QMDP is better than COMDP. MLQ itself does not perform as well as QMDP, but still has a good performance on Jillesville. It can be concluded that the POMDP problem can have a complete solution for this domain.

There is also the comparison between the normal MLQ and QMDP decision making algorithms with COMDP learning of the model and their respective MLQ learnt counterparts. As can be seen in this figure the MLQ learnt models work slightly less than their standard completely observable counterparts. Note that the performance loss is minor when keeping in mind that now the learning does not need direct state space access.

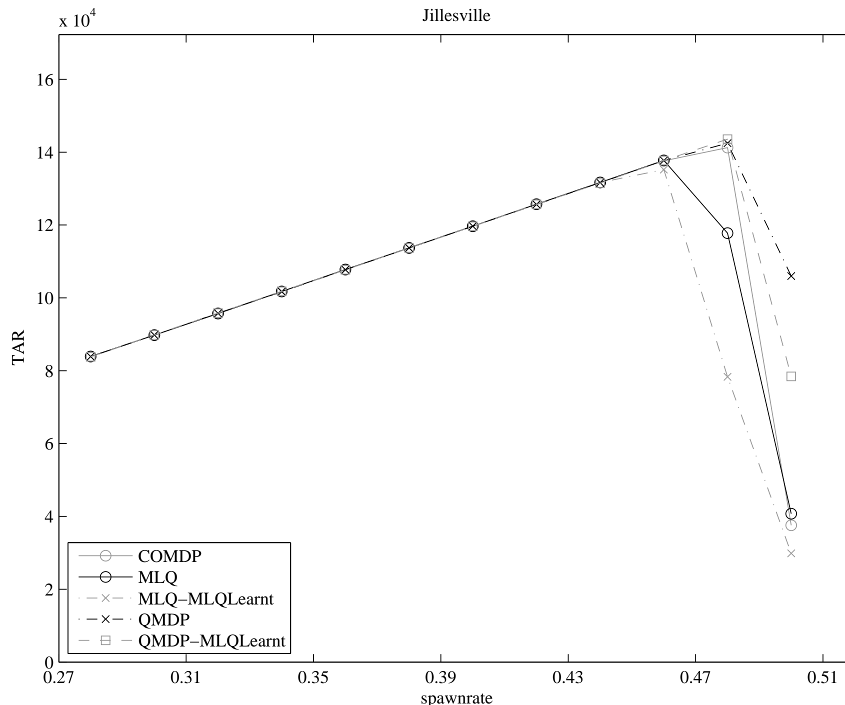


Figure 16: Result of Experiment 2a

Same test but now on the LongestRoad traffic network, see figure 17. It has a similar result as the test performed in test 1b. Again COMDP proves to be working worse than the MLQ and QMDP, like it was shown in Experiment 1b on the same traffic network, but now with methods that are implemented with MLQ learning of the model.

When looking at the comparison between the MLQ learnt POMDP methods and the COMDP learnt POMDP methods it is evident that the used traffic network and the size of the road do not have a real impact on the learning of the model. Moreover in this situation the learning generalization has a positive effect on the performance of QMDP, hence QMDP-MLQ learnt performs slightly better.

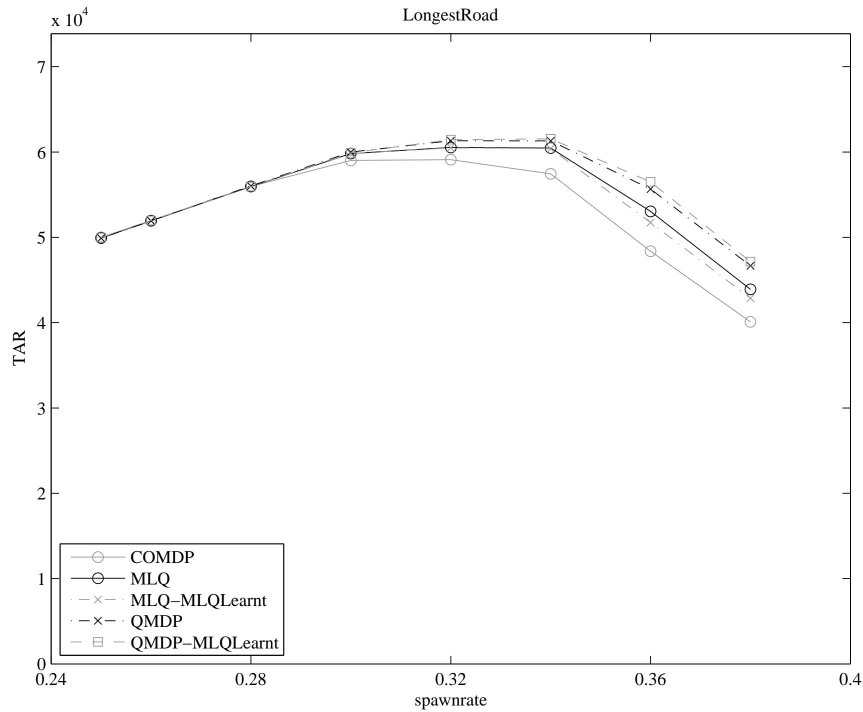


Figure 17: Result of Experiment 2b

8.3 Experiments 3 & 4: Sensor Errors

After the performance experiment of the learning of the model and the algorithms that use partial observability, it is interesting to see what effect sensor error would have on the performance. Previously was assumed that the sensors themselves work optimally and have a 100% flawless detection, $P(z|s) = 1$ if the road user was on that sensor. It can be seen in figure 18 that the used sensor performance is: no error, $P(z|s) = 1$ for QMDP, 30% error, $P(z|s) = 0.7$ for QMDP-error0.3, 60% error, $P(z|s) = 0.4$ for QMDP-error0.6 and 90% error, $P(z|s) = 0.1$ for QMDP-error0.9. The experiment proves that the performance goes down with more sensor error, but the impact might not be as high as could be imagined.

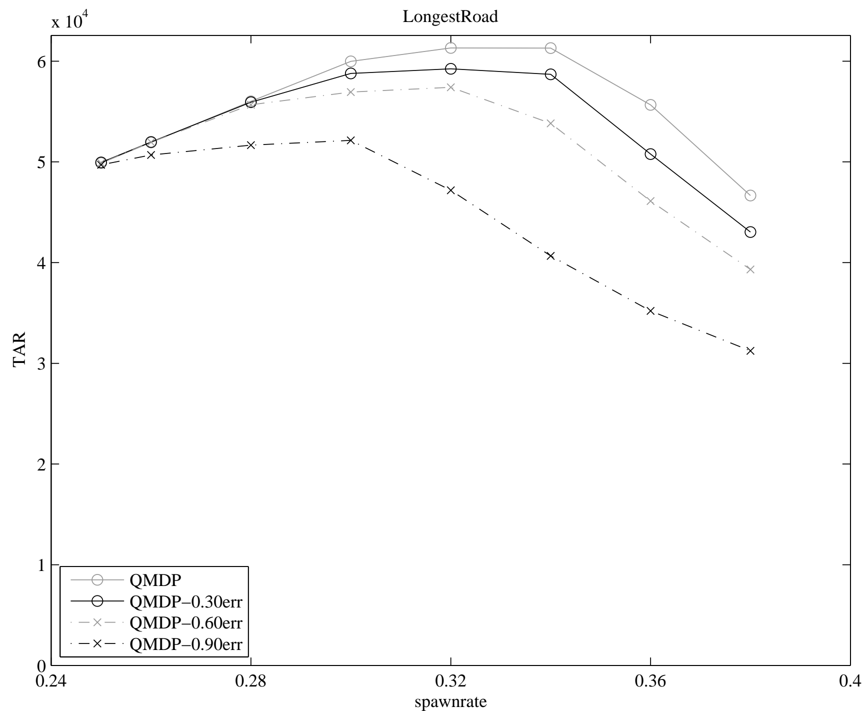


Figure 18: Result of Experiment 3

Figure 19 shows a similar experiment as experiment 3, but now featuring MLQ instead of QMDP. The biggest difference here is that MLQ appears to be more resilient to noise than QMDP, since the situation with the lowest sensor errors perform relatively the same. QMDP on the other hand had almost equal spread in performance between the different sensor error levels. QMDP weighs a lot more than MLQ in order to make a decision, so the influence of not detecting a certain road user will have a much larger impact on QMDP decision making than that of MLQ. For both the 0.9 error perform significantly worse than the others. This shows the need for sensors is still there even for MLQ.

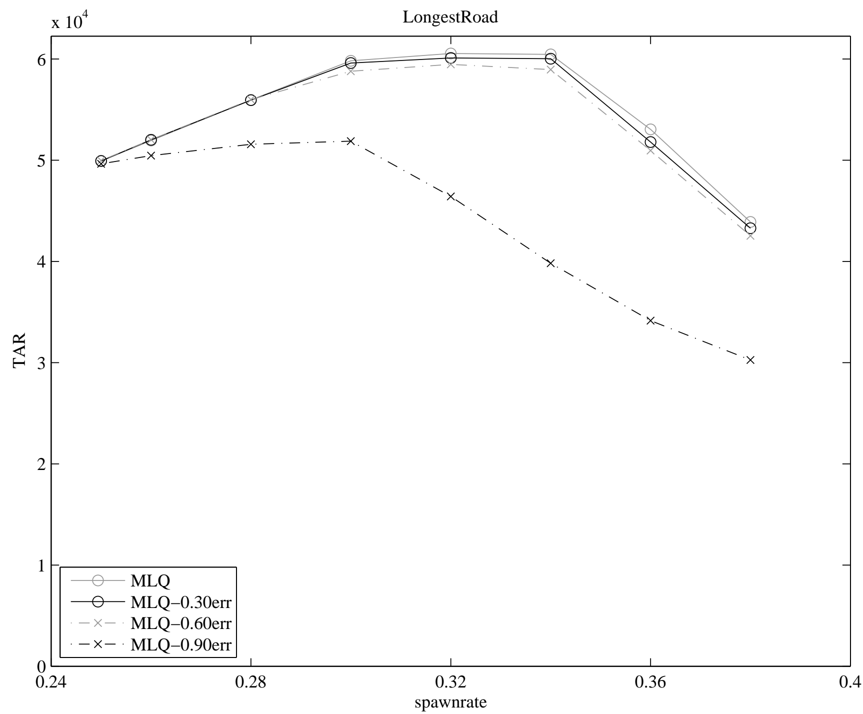


Figure 19: Result of Experiment 4

8.4 Experiment 5: Gaussian vs Uniform road user noise methods

The experiment in this section is meant to compare different assumptions made about road user movement in the model, figure 20. When assuming that the road users change speed with a probability distribution that is Gaussian the performance is worse than if the assumption is made that the road users change speed with an uniform probability distribution for both MDP and QMDP. Since the speed change probability distribution for the real road user is neither Gaussian nor Uniform it seems it is better to take a naive uniform assumption than a wrong assumption.

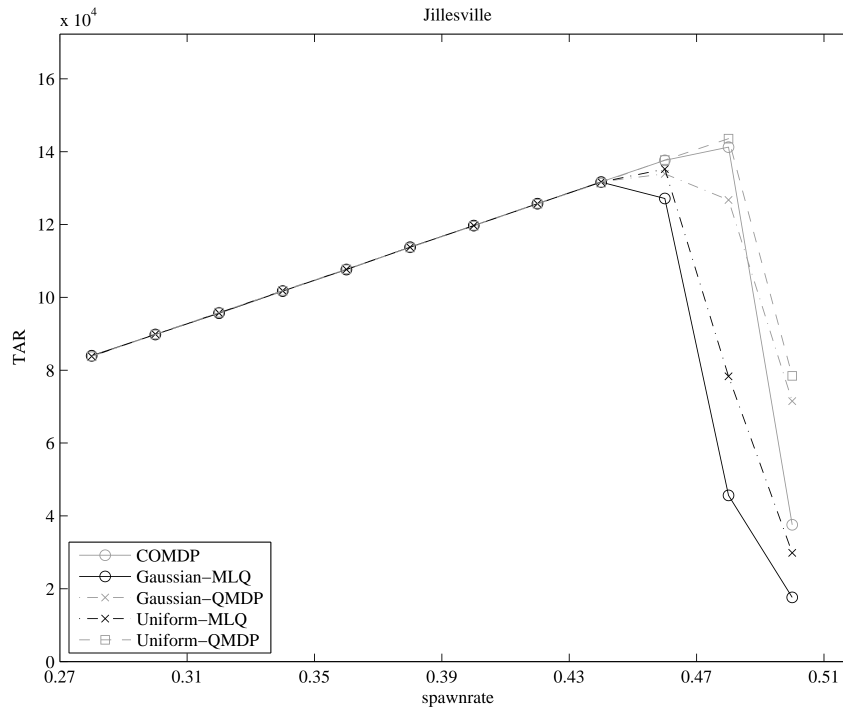


Figure 20: Result of Test 5

8.5 Experiment 6: Explaining COMDP performance

In the last sections COMDP performed slightly less than expected. Because it is the only algorithm used in these experiments that has access to the real state information the results were expected to be the higher than with the POMDP algorithms. Since the performance decreased when the road became longer the suspicion arose that it was due to the lower counts of a road user passing through a certain state on the drive lane. These counts were called the support ϕ for the state transition probabilities. Experiment 6 was a way to exclude the possibility of under-performance by COMDP compared to QMDP due to the support ϕ of the individual transition probabilities of the model. Because there are many different transition probabilities on a completely observable domain the individual support ϕ for each transition can be low. QMDP uses a mixture of these low supported transitions in gain computation to overall have a better supported policy. Limiting the amount of states in the state space for each road the support ϕ for each individual state transition will generally be higher. This was observed between Jillesville and LongestRoad where Jillesville has significantly more states than LongestRoad, while the number of states per lane is much lower. In order to get insight into this theory an infrastructure is needed with as little roads as LongestRoad, but with the lowest amount of states per lane road possible, see figure 13. The Simple infrastructure has the smallest grid possible in the simulator eliminating the chance of a state not being visited by road users.

The resulting experiment in figure 21 shows that the performance of COMDP is equal to that of QMDP and MLQ. This provides support for the theory about the support ϕ.

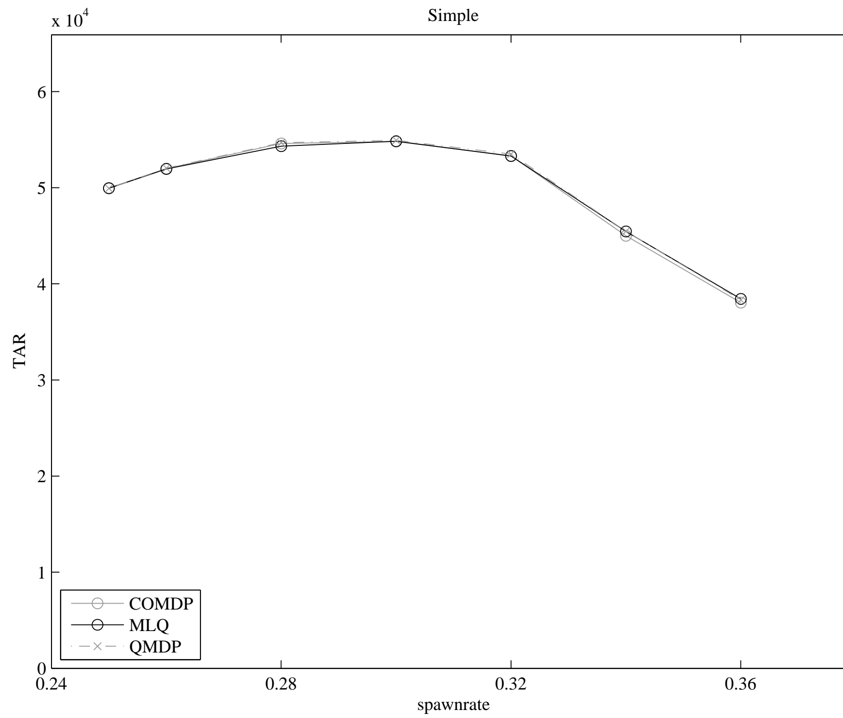


Figure 21: Result of Test 6

8.6 Experiment 7: Improvements for COMDP?

Experiment 6 suggested that the lack of support ϕ could be the reason for the lower performance of COMDP compared to QMDP.

Normally for COMDP the model is initiated fresh with every probability. $Q(s,a)$ state-action value pair is initialized to zero. The actions that are performed with these $Q(s,a)$ will not be guaranteed to be optimal, the policies can actually be pretty bad. While the simulator keeps running, more and more transitions are counted and $Q(s,a)$ pairs are updated. When the simulator can run long enough without causing the infrastructure goes into a deadlock, eventually the policies A_n^{opt} from equation 22 will approach the optimum. When spawn rates are high however, the strain on the network is so high, that every non-optimal or really bad policy can easily lead the network to a deadlock.

As discussed in section 3, with the heuristic POMDP approximation, QMDP gain computations are based on the underlying Completely Observed MDP multiplied with the belief state probabilities over the entire state space. In the Completely Observable domain a similar thing like taking multiple states can be done without the use of the belief state.

A mixture of states can prevent the use of a state that has not enough support ϕ for the calculation of the gain. Instead the neighbour states can be used in the case their support ϕ is higher. If the support ϕ for a state is higher, the value of the Q(s,a) is more representative. A bad value is really bad and a good one really good.

To try out the impact of using mixtures of states an algorithm is defined. Keep in mind that states can be seen as positions on a road and that a mixture of states can be seen as a region on a road. The algorithm that implements this change to COMDP is called Nearest Neighbour COMDP, or NN-COMDP and is defined as follows:

$$Q(s_p, a) = \begin{cases} \sum_k \widehat{\phi}(\lambda_k, \Phi) Q(\lambda_k, a) & \text{if } \sum \phi(\lambda) > \phi(s_p) \\ Q(s_p, a) & \text{otherwise} \end{cases} \tag{33}$$

where $\widehat{\phi}(s, \Phi) = \frac{\phi(s)}{\sum_{\lambda \in \Phi} \phi(\lambda)}$ and $\Phi = \{\lambda_0..\lambda_k\}$ and λ are the k surrounding states around s_p excluding s_p.

The surrounding states will only be used in gain computation if their probabilities are together higher than the probability of the centre state.

In figure 22 and figure 23 the results are shown for the NN-COMDP, with two nearest neighbours used. Figure 22 shows a big improvement for COMDP with the NN-COMDP. It even outperforms QMDP on the Jillesville infrastructure. Apparently it is possible to improve COMDP, which means COMDP cannot be seen as the optimal method if the states are fully observable. The exception where QMDP is better than NN-COMDP is if the model is learnt by using MLQ. Learning the model with MLQ gives a more generalized model. The reason therefore is that not only the gain computation uses the preprocessed road user movements, but also the learning of the model, which provide even more support ϕ and better learnt values for the most important states.

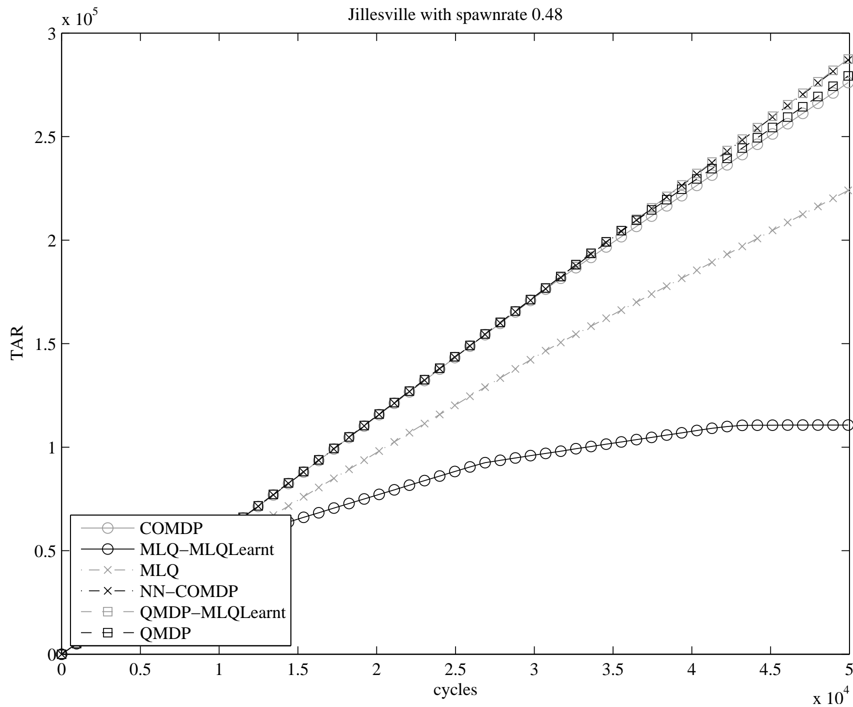


Figure 22: Result of Experiment 7a

If the test is performed on the LongestRoad infrastructure, as can be seen in figure 23, it does not give the same performance improvement. NN-COMDP is now performing equal to the normal COMDP and thus worse than QMDP. An explanation for this under performance might be that the amount of neighbours that are taken into account is too small, since the road is much larger. Moreover the arbitrary amount of neighbours chosen seem not reliable enough to make a real improvement on COMDP, but the potential is there. QMDP uses a better way to spread by use of the belief state properties, causing amounts of neighbours that are used to be more adaptive to the situation and not fixed.

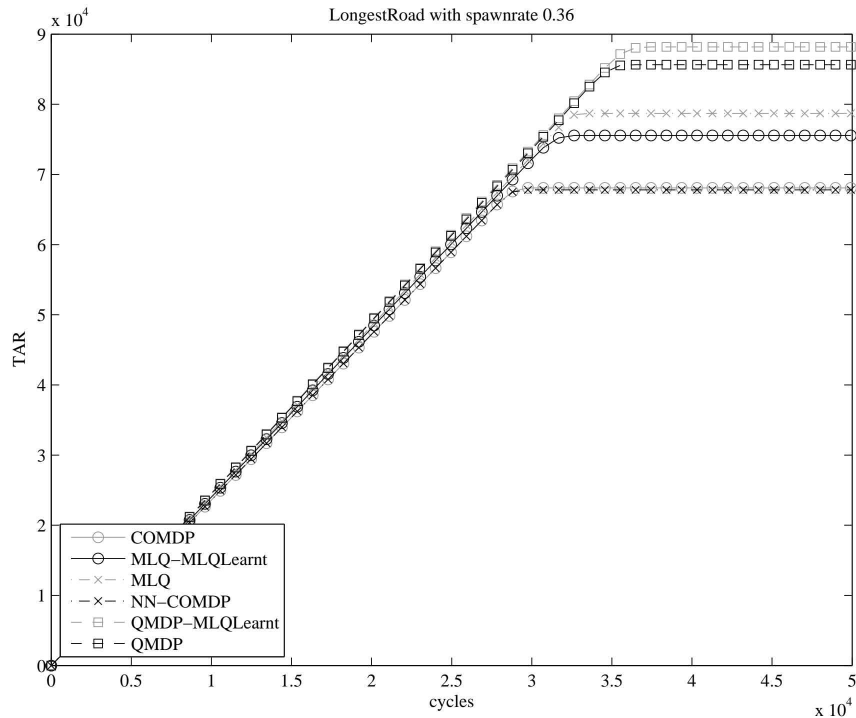


Figure 23: Result of Experiment 7b

9 Conclusion & Future work

The goal of finding a way to cope with the partial observability problem was successfully reached. When using the primitive POMDP algorithm AIF. It was clear that the assumption that all the road users jumped to the front of the lane after they were observed by the first sensor on the road, is bad. A large improvement has been seen with implementation of MLS. This was because of using a belief state of the individual road users. Though MLS was still too primitive in that road users could be estimated on top of each other and that there were no assumptions of road users not being able to jump over each other. Incorporating this Multi-Agent approach in the beliefstate representation of an entire lane, the combined road user beliefstate provided the basis for MLQ and QMDP. These two methods outperformed MLS by far.

The two infrastructures mainly used, Jillesville and LongestRoad, provided a very difficult partially observable task. Jillesville had a large state space of all the roads combined which had to be broken down to representations of the lanes themselves. LongestRoad with only four roads but with a much larger number of states per road is in fact in a way a much bigger challenge, since these states could not be broken down. The size of the beliefstate representation was getting an issue when using QMDP. However using an approximation by removing beliefstate points that had the lowest probability mass provided a way to handle this without impacting the performance too much.

Using noisy sensors put a larger strain on QMDP than on MLQ. However both methods were still performing reasonably well under even extreme cases. Most of the time some quality can be expected from sensors, so this would not pose an issue for real-world application deployment.

The beliefstate was estimated by the use of a predefined road user movement model that predicted the changes of speeds road users had between time steps. The two tested models were the uniform and Gaussian distributed random models for changing speeds. The real random distribution, which only the COMDP method could observe, was neither uniform or Gaussian. MLQ and QMDP that are using a beliefstate with uniform road user model perform better than the same methods using the Gaussian road user model.

Overall the performance of the most advanced partially observable methods in this thesis, MLQ and QMDP, perform equal or better than the existing COMDP method. These results where above expectations, because these were heuristic approximations and supposedly more sub-optimal than COMDP. Also the learning of the model with the use of MLQ improved the results even more. Besides finding a solution for partial observability there was also another side-effect. Generalizing a model on large state spaces seemed to improve when the beliefstate patterns were used instead of the

real locations of the road user. This way of pre-processing showed a dramatic improvement on the LongestRoad infrastructure. This actually gave suggestions on how to improve the completely observable model.

Improving the completely observable model was not in the scope of this thesis. However it is useful for future research and for further understanding of the state transition probability support ϕ of the model. Possibly a similar pre-processing can be done, like QMDP, on COMDP to further improve the performance.

It would also be interesting to test the POMDP algorithms on real world applications to see how the conversion of the continuous roads to a discrete model has an impact on the performance. Since adding more states to the road makes the problem harder to cope with by means of computations, the skipping of belief states provide a good way to handle it. If it still poses too big of a computational problem MLQ can be used which uses the least amount possible. The current real world implantation of traffic lights is not very advanced, QMDP and also MLQ would be a significant improvement.

References

[1] Cassandra, *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes* PhD thesis, Brown University, 1998.

[2] M. Hauskrecht. *Value-function approximations for partially observable markov decision processes.* J. of AI Research, Vol. 13, p. 33-94, 2000.

[3] T. Jaakkola, S. P. Singh and M. I. Jordan *Monte-carlo reinforcement learning in non-Markovian decision problems*, Advances in Neural Information Processing Systems 7, 1995.

[4] L. P. Kaelbling, M. L. Littman, and A. W. Moore. *Reinforcement learning: A survey.* Journal of Artificial Intelligence Research, 4:237-285, 1996.

[5] L. P. Kaelbling,M. L. Littman and A. R. Cassandra. *Planning and acting in partially observable stochastic domains.* Artificial Intelligence, Vol. 101(1-2), p. 99-134, 1998.

[6] M. L. Littman, *Memoryless Policies: Theoretically limitations and practical results.* From Animals to Animats 3, Brighton, UK, 1994

[7] J. Liu and R. Chen. *Sequential Monte Carlo methods for dynamic systems.* Journal of the American Statistical Association, 93, 1998.

[8] T. M. Mitchell *Machine learning.* New York: McGraw-Hill, 1997.

[9] N. L. Nihan, X. Zhang and Y. Wang *Evaluation of Dual-Loop Data Accuracy Using Video Ground Truth Data*, 2002.

[10] J. Pineau, G. Gordon, and S. Thrun. *Point-based value iteration: An anytime algorithm for POMDPs.* In Proceedings of the 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico, IJCAI, 2003.

[11] R. D. Smallwood and E. J. Sondik *The optimal control of partially observable Markov processes over a finite horizon.* Operations Research 21, p. 1071-1088, 1973

[12] M. T. J. Spaan and N. Vlassis. *A point-based POMDP algorithm for robot planning.* In Proceedings of 2004 IEEE International Conference on Robotics and Automation (ICRA), 2004.

[13] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* MIT Press, Cambridge, MA, 1998.

[14] R. S. Sutton. *Integrated architectures for learning, planning, and reacting based on approximating dynamic programming.* In Proceedings of the Seventh International Conference on Machine Learning, Morgan Kaufmann, Austin, TX, 1990.

[15] M. Steingröver, R. Schouten, S. Peelen, E. Nijhuis, and B. Bakker. *Reinforcement learning of traffic light controllers adapting to traffic congestion.* In Proceedings of the Belgium-Netherlands Artificial Intelligence Conference, BNAIC05, 2005.

[16] C. T. Striebel, *Sufficient Statistics in the Optimal Control of. Stochastic Systems*, Journal of Mathematical Analysis and Applications, Vol. 12, p. 576-592, 1965.

[17] G. Tesauro. *TD-gammon, a self-teaching backgammon program, achieves master-level play.* Neural Computation, Vol. 6(2), p. 215-219, 1994.

[18] T. L. Thorpe and C. Anderson. *Traffic light control using Sarsa with three state representations.* Technical report, IBM Cooperation, 1996.

[19] S. Thrun *Particle Filters in Robotics* In Proceedings of Uncertainty in AI (UAI) 2002.

[20] C. Watkins, *Learning from Delayed rewards* Ph.D. dissertation. Kings College, Cambridge, England, 1989.

[21] M. Wiering, J. van Veenen, J. Vreeken, and A. Koopman. *Intelligent traffic light control.* Technical report, Dept. of Information and Computing Sciences, Universiteit Utrecht, 2004.

[22] M. Wiering, J. Vreeken, J. van Veenen, and A. Koopman. *Simulation and optimization of traffic in a city.* In IEEE Intelligent Vehicles symposium (IV'04), 2004.

[23] C. C. White, W. T. Scherer. *Finite memory suboptimal design for partially observed Markov decision processes.* Operations Research, Vol. 42(3), p. 439-455, 1994.

A Work distribution

A.1 Written sections

Although all sections are revised by both authors, the first revision of the individual chapters were written by:

- Section 1: Roelant Schouten.
- Section 2: Roelant Schouten.
- Section 3: Merlijn Steingröver
- Section 4: Roelant Schouten.
- Section 5: Merlijn Steingröver
- Section 6.1-6.2: Merlijn Steingröver
- Section 6.3-6.4: Roelant Schouten
- Section 6.3-6.5: Merlijn Steingröver
- Section 7: Roelant Schouten.
- Section 8.1-8.3: Roelant Schouten.
- Section 8.4-8.6: Merlijn Steingröver.
- Section 9: Merlijn Steingröver.

A.2 Programmed parts

Many parts were implemented on the GLD software. The following list expresses main implementations, most of the elements have been implemented by both authors, however the time spent by refining the implementation and the bug fixing was primarily done by:

- Implementation road users being able to change speeds: Roelant Schouten
- Partial observable drive lanes added to the system and infrastructures: Merlijn Steingröver
- Implementation of the road user beliefstate: Roelant Schouten
- Implementation of the drive lane beliefstate: Merlijn Steingröver
- Adding Gui elements for Partial observable lanes: Merlijn Steingröver
- Implementation of the Uniform and Gaussian noise methods: Roelant Schouten

- Spawnrate calibration implementation: Roelant Schouten
- Implementation of MLS: Roelant Schouten
- Implementation of MLQ: Roelant Schouten, Merlijn Steingröver
- Implementation of QMDP: Roelant Schouten
- Implementation of the model learning with MLQ: Merlijn Steingröver
- Beliefstate approximation skiplist algorithm implementation: Merlijn Steingröver
- Implementation of NN-COMDP: Merlijn Steingröver