Using Reinforcement Learning for Traffic Lights Control

School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia

COMP9417, Machine Learning, Assignment 3

Han Li
hli@cse.unsw.edu.au
3299976

Lawrence Yao
lawry@cse.unsw.edu.au
2252726

Vaughan Rouesnel
vjro855@cse.unsw.edu.au
3188927

1. INTRODUCTION

Reinforcement learning is naturally suited to control problems since it readily fits the idea of agent and its environment and in particular, learning on the job. Unlike classification algorithms, it is not necessary to have a training set available. In this assignment, we look at the problem of an agent controlling a traffic light with three signals (red, amber, green) at a four roads intersection (north-south, south-north, east-west, and west-east). We implement various algorithms and state representation, and compare their performance using various parameters.

2. DEFINITION

2.1 Traffic Model

The traffic is flowing in two ways on two intersecting roads. Based on the direction of car flows, the traffic model is comprised of four independent one-way flows, denoted as West-East, East-West, North-South and South-North. For example, in the North-South flow, all the cars travel in one direction, that is, from north to south.

The competing flows of the traffic are controlled by a set of traffic lights. As there are only two intersecting main roads (regardless the directions), two mutually exclusive traffic lights are sufficient. They are denoted as lightWE (which controls West-East and East-West), and lightNS, for North-South and South-North traffic flows. There are three signals, namely RED, AMBER and GREEN. A car should stop if there is an AMBER or RED signal, and the cars behind it form a stationary queue. In addition, the lights can not change consecutively in less than three timesteps.

Each flow is modelled as a standalone road. The road is 100 units in length (from 0 to 99), and intersects with the roads in the perpendicular directions at Unit 49 and 50. The traffic light is positioned at road intersections. At each timestep, one car enters the road at Position 0, with the probability that equals to the traffic intensity (e.g. 15%). Cars are removed from the road when they reach the boundary (i.e., beyond unit 99).

Assignment 2 June 5, 2011, Machine Learning, COMP9417

2.2 State Representation

The traffic state consists of the light's state and the car's state. As the light signals are mutually exclusive, there are only four possible states, namely (GREEN, RED), (AMBER, RED), (RED, GREEN) and (RED, AMBER), denoted as 0, 1, 2 and 3 respectively. Moreover, the value of light delay since last change is 0-3. Hence, the state space of lights is 4 x 4.

However, defining the car's state is less straightforward. The occupancy of each unit by a car can be marked as 0 or 1. As there are 48 units before the traffic lights on each road, the state space of cars can be as large as 2^{192}. Therefore, we propose three variants of state representation to maintain the space in a feasible size.

2.2.1 *Default State*

The Default State is described in the assignment specification. On each road, only the position of the closest car from the intersection is counted, and only the first 9 units from the intersection are inspected. Therefore, the position is between 0 and 8 inclusive, and 9 denotes no cars. The space size of Default State is 10^4.

2.2.2 *JamNess State*

The JamNess State is a representation of how busy the traffic is on that road (i.e. traffic jam-ness). Each car is given a weighting based on their distance from the intersection. Cars closer to the intersection have a greater weight than cars further away from the intersection. To calculate JamNess, we introduce an importance value of each car, denoted as 0.5^i, where i is the position of the car from the traffic lights (at Unit 49). For example, the importance of the car at Unit 46 is 0.5^3, since it is 3 positions away from the lights. The sum of all the importance values on one road is between 0 and 0.9999999. Then, the JamNess is calculated as $sum * 10$ and rounded down to an integer, which is between 0 and 9 inclusive. The space size of JamNess State is 10^4.

2.2.3 *Occupancy State*

The Occupancy State represents the presence of cars at each unit. As discussed above, each unit is marked as 0 or 1, there are 2^{48} states if all the units are counted. However, since the traffic lights can be switched every three timesteps, we can speculate that the closest three or four positions from the intersection are more important than the rest. Hence, we define two states, Occ_3 and Occ_4, which are represented by a 3-bit and 4-bit integer respectively, indicating the occupancy of the closest three or four positions. For example, a binary value 100 in Occ_3 indicates the first position is occupied, while the second and the third are empty. A binary value 0101 in Occ_4 means the second and the fourth positions are occupied, whilst the rest are not. Since there are four roads, the space size of Occ_3 is 2^{12}, and that of Occ_4 is 2^{16}.

2.2.4 Density State

The Density State records the number of cars on the N closest positions from the intersection, where N is 3 in the implementation, since the traffic lights can only be switched every three timesteps. The number of cars in the closest three positions is between 0 and 3, so the space size of four roads is 4^4 or 2^8.

There is an extension of Density State (N=3). To help the controller predict the future traffic, we not only calculate the number of cars at the first three positions (i.e., from Unit 48 to 46), we also calculate the car number of the second three positions (i.e., from Unit 45 to 43). This extension is called Density_6, as there are six positions involved. By constrast, Density State (N=3) is called Density_3. The space size of this extension is $(4 * 4)^4$, or 2^{16}.

3. IMPLEMENTATION

3.1 Learning Algorithms

Reinforcement learning is concerned with how to learn a control policy (a mapping from the states of environment to control actions) so as to maximise a cumulative reward signal. Let s_t be the state of the server at time t. The controller selects an action a_t from a finite set A resulting in the system transitioning to state s_{t+1} at time $t+1$. The controller earns a reward r_{t+1} for this transition. We implemented Sarsa, Q Learning and a OneRule algorithm as the benchmark.

3.1.1 Sarsa

This algorithm is based on the following formula from Sutton [1] on page 145.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{1}$$

where α is the learning rate, and γ is the discount factor. Some important implementation details are:

1. The 3 timesteps immediately after a light switch is not involved in learning (i.e. Q is not updated in those 3 timesteps), however the reward is accumulated during the 3 timesteps;
2. Sarsa uses the ϵ-greedy policy, so the best action is picked with probability $(1-\epsilon)$ and a random action is picked with probability ϵ. Picking a random action means the light may switch with 50% probability;
3. The ϵ is not discounted over time, although Sutton [1] mentions that Sarsa converges with probability 1 to an optimal policy only if the policy converges in the limit to the greedy policy (which requires ϵ to converge to 0 in the limit). The reason why ϵ is not discounted is so it can be compared directly with QLearningBasic algorithm using the benchmark settings.

3.1.2 Q Learning

In Q-Learning [2], the learning agent calculates the quality (or Q-value) of a state-action combination, denoted by $Q(s, a)$, from its interaction with the environment using the formula:

$$\begin{aligned} Q_t(s_t, a_t) = Q_t(s_t, a_t) + \\ \alpha[r_{t+1}) + \gamma max Q_t(s_{t+1}, a) - Q_t(s_t, a_t)] \end{aligned} \tag{2}$$

where $max Q_t(s_{t+1}, a)$ is the maximum Q value of s_{t+1}.

The difference between Sarsa and Q Learning is that Sarsa picks the next action according to the ϵ-greedy policy and uses the same action to update the Q values, whereas Q Learning picks the action that gives the best Q value for Q value updates, but uses an exploratory action in execution. In other words, the next action to be executed can be different from the action used to update the Q value. Therefore, Sarsa is on-policy, while Q Learning is off-policy.

3.1.3 OneRule

Some would argue that this is not a true "predictive" algorithm, more of a "reactive" algorithm. This algorithm encodes a single rule (which is hard-coded in the source): the rule is to change traffic light for a road whenever there is a certain number of cars waiting on that road. This certain number of cars or *threshold* is fixed at 1 in our experiments. Whenever a road has this threshold number of cars waiting, the traffic signal will change (pending it has waited for 3 time steps already). Assuming a road full of cars (except for the intersections) and a threshold of 1, this results in an algorithm that lets 3 cars go past north-south (and south-north), then alternates to let 3 cars go past east-west (and west-east), then alternates to let 3 cars go past north-south (and south-north) and so on.

Since this is not a reinforcement learning algorithm, it is used for comparison purposes only.

3.2 Refinements

Our refinements aim at two goals: the first goal is to accelerate the learning process by different Q value update strategies; the second goal is to improve the learning outcomes, that is to reduce the average waiting time of the cars.

3.2.1 Eligibility Traces

Eligibility traces are one of the basic mechanisms of reinforcement learning for accelerating the learning process. An eligibility trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action. The trace marks the memory parameters associated with the event as eligible for undergoing learning changes.

When applying eligibility traces, a problem occurs for off-policy methods such as Q Learning when exploratory actions occur, since we backup over a non-greedy policy. In the implementation, we use Watkins' approach. That is, zero out eligibility trace after a non-greedy action, and do max when backing up at first non-greedy choice.

3.2.2 Boltzmann Distribution

We have adopted the policy of drawing the action with probability p from the Boltzmann distribution [2]:

$$p(s, a) = \frac{e^{Q_t(s,a)/\tau}}{\sum_{a^* \in A} e^{Q_t(s,a^*)/\tau}} \quad (3)$$

where $p(s, a)$ is the probability of selecting action a in state s, and τ is a temperature parameter that tunes the randomness of selecting the actions. τ is an annealing factor that is decreased in every iteration thereby reducing randomness as well. When τ approaches 0, the algorithm becomes a greedy algorithm where the action with highest value is always chosen.

By comparison with ϵ-greedy, the advantage of this Boltzmann distribution policy is that, the probability of choosing an action depends on its Q value. The larger the Q value is, the bigger the chance it gets selected. In ϵ-greedy policy, however, the action with the maximum Q value always gets selected with probability $1 - \epsilon$.

4. EXPERIMENTS

The experiments are intended to study the effects of the following variables on the reinformance learning performance:

1. Different state representations. Six repsentations are implemented: Default, JamNess, Occ_3, Occ_4, Density_3 and Density_6.

2. Different learning algorithms, including different exploration strategies and the effects of eligibility traces. Six implementations are to be compared: OneRule, Sarsa, Q Learning Basic with ϵ-greedy policy, Q Learning with Boltzmann distribution policy, Q Learning with eligibility traces update, and the one called Ultimate, with the best refinements we can make.

3. Varied reinformance learning parameters, such as learning rate and discount factor;

4. Different traffic intensities.

4.1 Benchmark Set-up

To compare so many variables, it is important to setup a benchmark for all the experiments. We designate a standard set-up which is called the "benchmark". This benchmark consists of the following reinforcement learning algorithm set-up

Algorithm QLearningBasic

State Representation DensityState3

Learning Rate $\alpha = 0.1$

Discount Factor $\gamma = 0.9$

Policy ϵ-greedy with no discounting, $\epsilon = 0.1$

Traffic Intensity 0.15, i.e. 15% of the road units have cars

In all of our experiments, we run the algorithm for 20 million (20,000,000) timesteps. At every 100,000 timesteps, we take a "snapshot" of the performance measure, which is the average number of cars waiting (or queued) per timestep in the last 100,000 timesteps. This average value is calculated as the total number of cars queued in the period divided by 100,000. We compare this averaged performance measure,

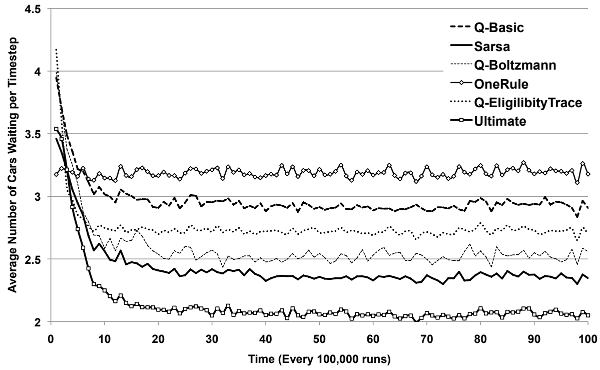

Figure 1: Best Algorithm

which is better if it is near 0 (very few cars waiting at every timestep), and worst if it is a large positive value (lots of cars waiting at every timestep).

In every experiment, we start with this benchmark set-up and vary a single parameter. We then run the algorithm once for a particular value of the varying parameter and record the performance snapshots. The same random number generator seed is used at every run, so the traffic pattern is stable across runs.

5. RESULTS AND DISCUSSION

Fig. 1 To determine the best choice of algorithm we ran an experiment using five configurations of algorithms and refinements whilst keeping the state representation, learning rate, discount factor and traffic intensity constant in accordance with the benchmark setup. We also ran the benchmark setup labelled as Q-Basic. The best algorithm without refinements running against the benchmark setup is the Sarsa algorithm converging to just under 2.5 waiting cars per timestep. The Q-Learning algorithm's performance is best when using Boltzmann's action selection policy and also benefits from the use of an eligibility trace but is still beaten by the Sarsa algorithm. The Ultimate algorithm is a modified version of the Sarsa algorithm using an action selection policy which chooses an action with probability p from the Boltzmann distribution rather than using the ϵ-greedy strategy, and includes the use of eligibility traces. The Ultimate algorithm delivers significantly better performance than all other experiements after 1,000,000 timesteps and converges with just above 2 cars waiting per timestep on average.

Fig. 2 From varying the eligibility decay factor over our benchmark setup, we observed the ideal decay to be set to 0.85 however it is evident that this factor alone has little impact on the performance of the overall algorithm.

Fig. 3 Perhaps unsurprisingly, as the traffic intensity increases from 15% to 25%, we found more cars waiting for a green light with an overall performance drop. Using Density_3 and Density_6 at 20% traffic intensity, we observed similar performance over 2,000,000 timesteps. However, Density_3 — having a smaller state space — starts to overfit the traffic patterns whereas

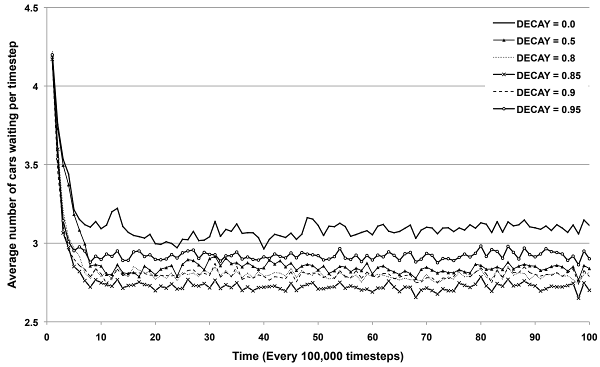

Figure 2: Eligibility Trace

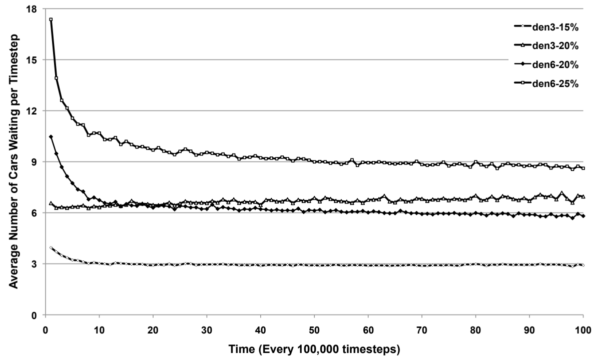

Figure 3: Varying Traffic Intensity

Density_6 — having a larger state space — continues to converge to optimal. Also, Density_6 takes longer to converge relative to Density_3, most likely due to the larger state space.

Fig. 4 In this experiement we vary the learning rate from 5% to 25% using the benchmark setup. It was observed that the optimal learning rates were between 5% and 10% evident after 2,000,000 timesteps. However, up until 1,000,000 timesteps a learning rate of 5% produces a significantly higher number of waiting cars than learning rates above 10%. Given the small difference between the number of waiting cars between 5% and 10%, we would consider a learning rate of 10% more optimal.

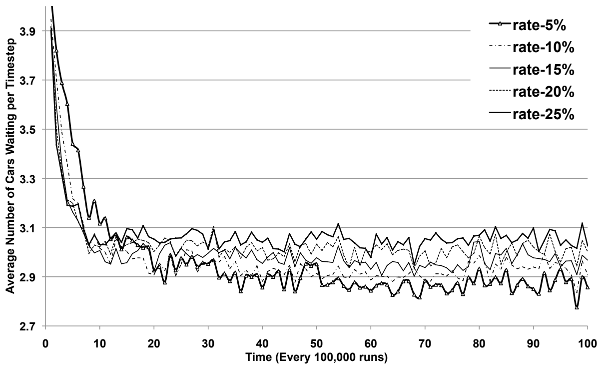

Figure 4: Varying Learning Rate

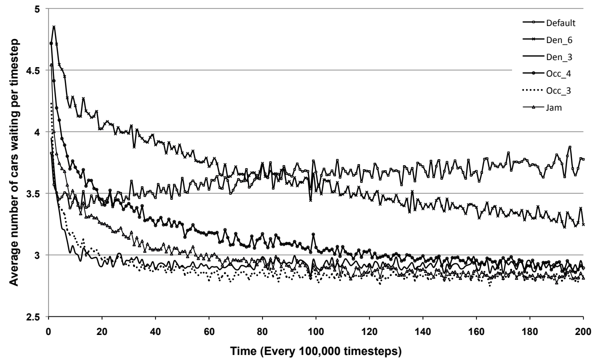

Figure 5: Varying State Representation

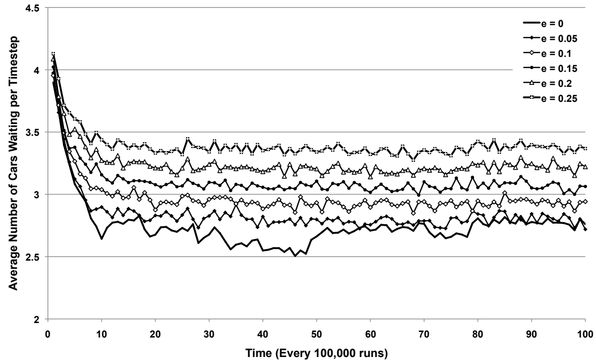

Figure 6: Varying the ϵ value

Fig. 5 This experiment uses the benchmark settings while varying the state space representation. The Default state produces the most sub-optimal performance after 2,000,000 timesteps. This representation provides quite a large state space but the type of information being conveyed is less useful than other representations as we only have information about one car on each road at any one time, giving no indication of how busy the traffic is which is the ultimate cause of accumulation of negative rewards. This is in contrast to the Occupancy, Density and JamNess spaces which provide a much better indication of how busy a road is in this sense. This is reflected in our experiment by the convergence of this group after 2,000,000 timesteps to between 2.7 to 3.0 average waiting cars. An interesting observation is the large difference between Density_6 and Density_3 spaces with Density_3 converging with the other best performing representations while Density_6 lags behind the group while narrowing in over the duration of the experiment. Occ_3 also performs better than Occ_4 initially. This is to be expected as it is logical that large state spaces will be slower to converge than smaller state spaces.

Fig. 6 This experiment uses the benchmark settings except for the ϵ value ('e' in the figure) which is different in each run. For example $e = 0.25$ means the QLearningBasic algorithm will explore 25% of the time. Note

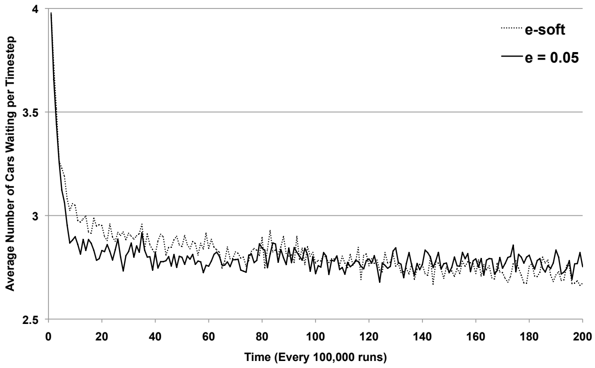

Figure 7: Discounting ϵ

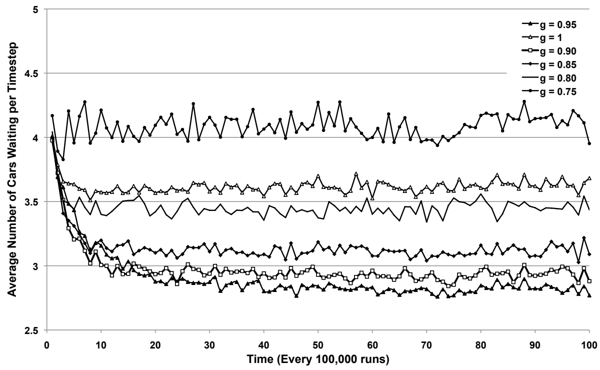

Figure 8: Varying the discount factor γ

the ϵ is not discounted over time. Better results are achieved with ϵ values near 0 but not 0, suggesting that exploratory actions that are not optimal are adding to the number of cars waiting; so the less exploratory actions taken the better. However, some exploration is required since $\epsilon = 0$ shows that overfitting occurs after about 4,600,000 timesteps. This overfitting is most likely due to the algorithm finding the optimal action-value function for the traffic pattern in the first 4.6 million timesteps, but the traffic patterns in the following timesteps are different and the optimal policy is no longer the most optimal.

Fig. 7 This figure shows the $\epsilon = 0.05$ result with a discounted ϵ run. The discounted ϵ run uses the benchmark settings (i.e. $\epsilon = 0.1$ initially) and discounting the ϵ by 0.9999999 per timestep (that's seven 9s). The result shows that a fixed ϵ stablises after about 4,300,000 timesteps without further improvement; but the ϵ-discounted algorithm converges to the optimal as time progresses. This matches with what Sutton [1] says about Sarsa converges with probability 1 to an optimal policy only if the policy converges in the limit to the greedy policy (i.e. ϵ tends to 0 in the limit); although this is QLearningBasic algorithm and not Sarsa.

Fig. 8 This experiment uses the benchmark settings except for the γ discount factor ('g' in the figure) which is different in each run. The results suggest the best discount factor to use is 0.95. As γ decreases from 0.95, the results gets progressively worse. Values larger than 0.95 should also get worse, although only one run with $\gamma = 1$.

6. CONCLUSION

Based on our experimental results, we have determined the optimal setup for implementing a complete learning algorithm for controlling traffic lights by choosing the best combination of space representation, algorithm, and refinements. We use a space representation of Density_3 as it has the smallest space size, yet delivers satisfactory performance (See Figure 5). We use the Sarsa algorithm, instead of the Q Learning algorithm for the main learning method as it delivers better performance than all experiments involving Q Learning (See Figure 1). For the action selection policy, we choose an action with probability p from the Boltzmann distribution, rather than using an ϵ-greedy approach (See Figure 1). Eligibility traces are used to accelerate the learning speed. The learning rate used is 0.1 with a discount factor of 0.9. The overall performance of this combination of parameters is demonstrated in the Ultimate algorithm in Figure 1.

The Ultimate algorithm provides the best performance over the long-run. It is a suitable means for traffic light control given that traffic lights have a relatively long lifespan allow a learning algorithm ample opportuntity to converge to an optimal state. It is worth noting that a much more cost effective option may involve a roundabout delivering savings in electricity, maintenance costs, and the need to write software to control traffic lights.

7. REFERENCES

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[2] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.