Solving Homogeneous Reinforcement Learning Problems with a Multi-Agent Approach

David Kauchak
Department of Computer Science
UC San Diego
La Jolla, CA 92093-0114
dkauchak@cs.ucsd.edu

Abstract

In this paper we examine reinforcement learning problems which consist of a set of homogeneous entities. These problems tend to have extremely large state spaces making standard approaches unattractive. We study lane change selection in a car traffic control problem as an example of such a problem. We show how a single agent problem can be translated into an approximating multi-agent problem. We provide learning results in a traffic simulator using this multi-agent approximation with Q-learning and R-learning. Learning in the multi-agent problem proceeds quickly and outperforms heuristic methods. Experimental results show that learned methods perform better than heuristic methods as traffic densities increase towards rush hour conditions. We summarize the translation method used from a single agent problem to a related multi-agent problem for car traffic control and propose this as a starting place for related problems.

1 Introduction

Reinforcement learning (RL) methods have been used to solve many problems where supervised methods are not appropriate. In many of these domains, the environment is too complicated to generate data for standard supervised methods. With reinforcement learning, an agent explores its environment, receiving a reward signal as it explores, and tries to optimize the sum of these rewards. There are many real world problems that can be formulated as reinforcement learning problems, but cannot be solved by current methods. In some situations, the problem may be too complicated for learning to converge in a reasonable amount of time. In other situations, restrictions posed by the learning setup make the methods inappropriate for practical use. In this paper we examine one such subset of RL problems, and propose a possible solution that uses previous learning methods, but modifies the problem setup. For an overview of RL methods and example applications, see Sutton and Barto (1998).

Not all single agent problems can be translated to multi-agent problems. Problems that are suitable contain an omnipotent agent that is controlling the behavior of a

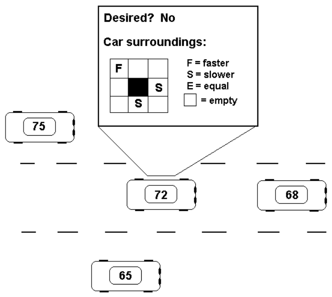


Figure 1: Example state of a single agent

number of similar entities. Examples of this include car traffic control, packet routing and game playing. In these domains the most intuitive rewards concern all of the entities. However, optimizing a system with tens of thousands of cars, routers or other entities is burdensome for current RL methods. In this paper, we propose a methodology to find solutions that approximate the optimal solutions that would be found if the original, single agent problem were solved. We provide experimental results for one example domain, car traffic control.

2 Learning Lane Changing Strategies

For the rest of the paper, we will examine car traffic control as an example of a reducible single agent problem. Reinforcement learning algorithms require three components to be defined: the set of states ***S***, the set of actions ***A*** and the reward function ***R(*s,a*)***, where **s** ϵ ***S*** and **a** ϵ ***A***. In this section, we define a straightforward single agent setup to the car traffic control problem. We identify problems with this setup and show how the single agent problem can be reduced to a multi-agent problem.

2.1 The Single Agent Setup

Before we investigate the multi-agent problem, we will first outline the idealized single agent case that we are attempting to approximate. The idealized version of the problem trains a single omnipotent agent. This agent controls the lane changing of all the cars. Each state **s** ϵ ***S*** consists of the positions of all the cars in the system, all cars' desired speeds and all cars' actual speeds. The actions **a** ϵ ***A*** consist of a lane choice for each car. The reward function, ***R(s,a)*** is the negative mean squared difference between a car's desired and actual speeds minus the number of lane changes made. Specifically:

$$R = -\sum_{x \in cars} (x(d) - x(v))^2 - 25 * \begin{Bmatrix} 1 \text{ if x changed lanes} \\ 0 \text{ otherwise} \end{Bmatrix}$$

where *x(d)* is the desired speed of the car, and x(v*)* is the actual speed. Notice that both of the parts of the reward function are over all cars. This reward function is calculated at **every** time step of the system. The 25 in front of the lane changes specifies how a lane change is valued versus the speed difference component. A constant of 25 means that a lane change is not desired until the speed difference gets over 5 mph. Note that the maximum reward possible for a state-action pair is zero and occurs when no lane changes are made and all cars are going their desired speeds.

Given this setup, any of the many standard reinforcement learning methods could be applied to get a solution. Unfortunately, the setup described above has a number of problems. If a small number of cars are to be modeled, then the state space may be of a manageable size. If the goal is to model normal traffic on a freeway, however, then the number of cars will be in the thousands. This large number of cars implies that the state space will be extremely large. This has two ramifications. We may be unable to represent the state space well because function approximators would have to be used. Worse, however, training such a large state space would take a large number of training runs. A more critical problem than the state space size is that the state space must consist of a static number of cars, determined prior to learning. For simulations this may be appropriate, however, for real world environments the number of cars will be constantly changing. The combination of these factors makes this setup unattractive.

2.2 Approximating the Idealized Problem

In this section we present an approximation to the idealized single agent setup. The basic idea is to reduce a single omnipotent agent problem to a multi-agent problem where each of the controllable entities in the single agent formulation becomes an agent. States and actions are changed to represent the states and actions available to one of these multi-agents. The main problem with this type of setup is that the reward function can no longer be calculated from a state and an action, but must be calculated from the states and actions of all the multi-agents. We provide a number of possible ways in which a new reward function can be calculated.

2.2.1 Actions

We will follow the approach of Moriarty & Langley (1998) in which the actions are whether to stay in the current lane, move left or move right (***A* = {move left**, **stay**, **move right}**). This set of actions provides a simplified action set and is independent of the number of lanes in which the cars are traveling. With this specification, the learning can be done with a different number of lanes than in the testing environment.

2.2.2 States

One of the crucial decisions for reinforcement learning is the state definition. Since this is a multi-agent system, the agent only has access to its surrounding environment and has minimal communication with the other agents. This is quite different from the single agent system that knows all the information about all the cars.

2.2.2.1 A Previous Approach to State Representation

Before we discuss the representation used in this paper, we first discuss one approach by Moriarty & Langley (1998) that we will improve upon. For the car being controlled, the state contains both the current and desired speeds. For the surrounding cars, the relative speeds and what type of lane changing policy is being followed are included (a binary choice between the learned strategy and a greedy, heuristic strategy). Except for the binary variables, all other variables are integers over a range based upon physical conditions (approximately 50 possible values). To simplify the state space, they chose to represent the location of the surrounding cars not by a numerical value, but by defining eight locations surrounding the car that could contain a car (front, rear, left, right, front right and left, and rear right and left). Although this state space provides a good representation of an agent's environment, the state space size is on the order of $50^{10} * 2^8 = 2.5 * 10^{19}$ states. This state space size has a number of consequences. Any state space this large must use some sort of function approximator for the value function. Also, many states will be visited infrequently, so some states will have to be inferred from similar states. Although these consequences are not necessarily bad, we hypothesize that this state representation is too detailed.

2.2.2.2 A More Concise State Representation

In this section we describe the state space used throughout the rest of the paper. We make two modifications to the previous representation. First, we do not include surrounding car types (greedy or learned). Although this information was useful for their experimentation, it is not worth the blow up in state space size. Second, Moriarty and Langley used integers to represent the other state variables. We hypothesize that this representation is too specific. How much information does a car gain by knowing that the car to its left is going 10 miles faster versus 11 miles faster? In this paper, we reduce the set used to represent adjacent cars to four values: faster, slower or equal to the car's actual speed or empty. The motivation for this choice was to minimize the set of possible values while still maintaining the relevant information. Further research is needed to investigate other possible choices, which could involve binning the relative speeds into speed ranges, such as "between 0-5 miles faster," etc.

As in Moriarty and Langley (1998), only the eight surrounding cars are used for position in the state representation. This specification is informative, but still concise. A surrounding space is considered empty if there is no car in that space within 200 ft. (this was chosen as a reasonable value, but could be set based on actual sensor performance).

Finally, instead of using both the desired speed and the actual speed of the car as state variables, we only consider whether the car is traveling at its desired speed or not[1]. The representation used in this paper is summarized in Figure 1. This construction gives a state space of size $4^8 * 2 = 131{,}072$ states, which is much smaller than the previous size of $2.5 * 10^{19}$ states.

Before we move onto describing the reward function it is worth examining this multi-agent state space size versus the idealized single agent state space. Consider modeling a system with 1000 cars. This is still much smaller than real world environments. Assume that the state space for the single agent consists of a simplified state for each car similar to that just described. For 1000 cars, there will be approximately $(131{,}072)^{1000} \approx 10^{5000}$ states. The multi-agent state space size is independent of the number of cars, but the single agent state space grows exponentially with the number of cars.

2.2.3 Reward function

The idealized, single agent reward function is in terms of all of the cars in the system. The multi-agent state, however, only represents the environment of a single car. We present two different methods for creating reward functions from the reward function for the single agent system.

The first function, which we call the SingleCar reward function, uses only the information directly available to one multi-agent and does not use information about the other agents in the system. Specifically:

$$r_1(x) = -(x(d) - x(v))^2 - 25 * \begin{Bmatrix} 1 \text{ if x changed lanes} \\ 0 \text{ otherwise} \end{Bmatrix}$$

[1] In the real world, a car could be going slower, faster or equal to the desired speed, but the simulation design prevents a car from going faster than its desired speed.

where, as above, *x(d)* is the desired speed and *x(a)* is the actual speed. This reward function relates directly to the single agent reward function, *R*, as follows:

In the traffic problem, the reward function for the multi-agent problem can be formulated similarly to the single agent problem, except that only the state of a single car is used. In some situations, however, the single agent reward function is not in a suitable form for this type of reduction. For example, if the single agent

$$R = \sum_{x \in cars} r_1(x)$$

reward function was the sum of the differences, squared, instead of the sum of the squared differences.

We propose a second, general purpose, method for approximating the single agent reward function in the multi-agent environment. The reward function is calculated over all multi-agent states as if the problem were a single agent problem. This results in a single number, which is the reward for the single agent state. That single reward is given as the reward for all of the multi-agent states for that time step. The advantage of this method is that it is applicable in any situation where a single agent setup has been reduced to a multi-agent setup. The disadvantage is that the reward function for the multi-agents becomes noisy. For the traffic problem, we call this reward function AllCars.

2.3 The Learning Algorithms

Two different reinforcement learning algorithms are used: Q-learning (Watkins, 1989) and a variation of R-learning (Singh, 1994). Since the state space was reduced in size and the problem is formulated as a multi-agent learning problem where all the agents update the same function, a lookup table was used to represent the value function. We chose to represent the value function as the action-value function, *Q(s,a)*, since a complete model of the environment is not readily available to an agent; an agent does not know the locations of surrounding cars for the next time step.

2.3.1 Q-learning

Q-learning is a standard reinforcement learning method that has proved successful in a wide range of domains (Sutton & Barto, 1998 *** possibly a better citation). We use the simplest version of Q-learning which is undiscounted, one-step Q-learning. The Q-function is updated by the following rule:

$$Q(s_t,a_t) \leftarrow (1-\beta) * Q(s_t,a_t) + \beta * \left(r_{t+1} + \max_{a_{t+1} \in A} Q(s_{t+1},a_{t+1})\right)$$

where r_{t+1} is the reward for taking action a_t in state s_t, β is the learning rate parameter and $\max_{a_{t+1} \in A} Q(s_{t+1},a_{t+1})$ is the largest Q value over all possible next actions a_{t+1} from the next state s_{t+1}.

2.3.2 Modified R-learning

R-learning is a method for undiscounted, average reward reinforcement learning problems. There are many domains where the task is cyclical, which results in returns $\boldsymbol{R_t}$ (cumulative rewards) that may be unbounded. For traditional reinforcement learning methods to work in these domains they are traditionally discounted. Discounting forces rewards on the horizon to diminish, causing the returns to converge. In many domains, such as this one, discounting future actions does not make sense.

Average reward reinforcement learning methods generally work by estimating the long term average of the system and updating the value function based on whether the reward received is better or worse than the estimated average. For an overview of undiscounted, average reward methods see Mahadevan (1996). R-learning is a average reward RL where the action value function ***R*(s** $\boldsymbol{R(s_t, a_t)}$ is learned (Schwartz, 1993). This R-function represents the average adjusted value of doing action $\mathbf{a_t}$ in state $\mathbf{s_t}$. We use a modified version of R-learning (Singh, 1994) where the estimated average reward $\boldsymbol{\rho}$ are updated as follows:

$$R(s_t, a_t) \leftarrow (1-\beta) * R(s_t, a_t) + \beta * \left(r_{t+1} + \max_{a_{t+1} \in A} Q(s_{t+1}, a_{t+1}) - \rho_t \right)$$

$$\rho_{t+1} \leftarrow (1-\alpha) * \rho_t + \alpha * (r_{t+1} - \rho_t)$$

where $\boldsymbol{r_{t+1}}$ is the reward for taking action a_t in state s_t, α is the learning rate for the estimated average reward, β is the learning rate for the R-function and $\max_{a_{t+1} \in A} R(s_{t+1}, a_{t+1})$ is the largest R value over all possible next actions a_{t+1} from the next state s_{t+1}.

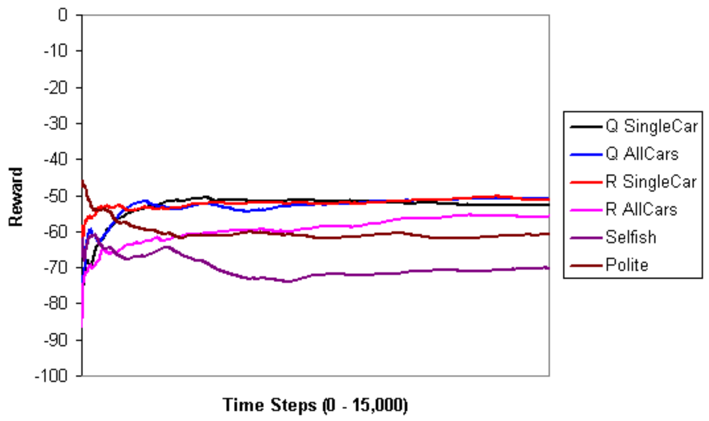
3 Experiments

We performed two sets of experiments to analyze the performance of the learning methods. Four learning methods were tested resulting from the product of the two learning methods (Q-learning and R-learning) and two reward functions (SingleCar and AllCars). Two heuristic methods similar to those used in Moriarty and Langley (1998) were used for comparison purposes. A greedy car changes lanes to the car if the car in front of it is going slower than it and the left lane is empty. If the left lane is not empty, but the right lane is empty then it will change right. A polite car implements the previous two rules, but also changes lanes to the right if the right lane is open and the car behind it is going faster than it.

All the learning methods used the same learning parameters, which were chosen based on values used by other authors: $\boldsymbol{\alpha}$=.05 and $\boldsymbol{\beta}$=.5 (Sutton & Barto, 1998; Mahadevan, 1996). The learning agents employ an ϵ-greedy decision strategy where the greedy action is chosen with probability (1- ϵ), otherwise an action is chosen randomly. $\boldsymbol{\epsilon}$ =.1 was used for all experiments.

3.1 Traffic Simulator Settings

Since learning in a real world environment is infeasible, a simulator was used. The simulator progresses by discrete time steps. Each time step is equivalent to one second. During a time step, all the cars will move forward and may change lanes. Lane changes occur in a single time step. This short lane change time may be unrealistic, however, difficulties arise in trying to simulate a multi-time step lane change.



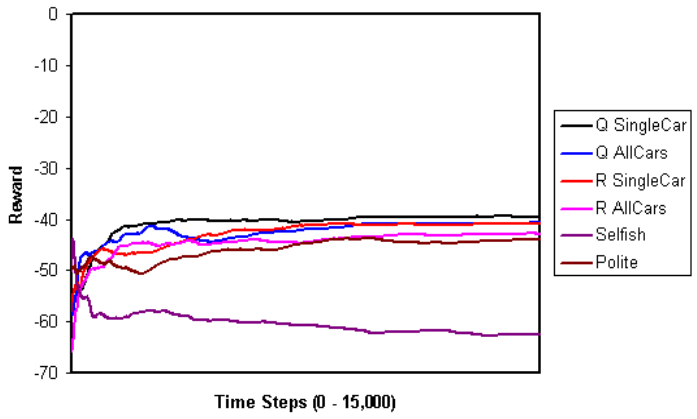


Figure 2: Average reward for the four learning methods and two heuristic methods (3 lanes on top and 4 lanes on bottom)

All cars in the simulator are the same size, 15 ft. This is a common size for a midsize car. The simulator consists of 5 miles of one way, multilane roadway. The road wraps around from the end to the beginning creating an infinite length of roadway. The desired speed of the cars is selected from a gaussian distribution with mean 72 mph and standard deviation 6. Given this distribution, most of the values should be between 66 mph and 78 mph with almost all values between 54 mph and 90 mph. These values appear consistent with observed freeway driving habits of California drivers.

One of the main difficulties in creating such a simulator is trying to model real human behavior in driving. There have been a few papers that have attempted to do this (Ehlert & Rothkrantz, 2001; Sukthankar et. al., 1997), however, implementing these would be a formidable task. For this paper, we model a simplified version of normal freeway driving. Future work will investigate more sophisticated models.

Cars accelerate and decelerate with two simple rules. Deceleration occurs at 2 mph/s. Acceleration is inversely dependent on the car's current speed:

$$A(v) = \frac{10}{\sqrt{v}} mph/s$$

where **v** is the current speed of the car. These rules were also used in Moriarty and Langley (1998). These rules are not intended to represent the maximum performance of a car. Instead, these rules attempt to model actual acceleration and

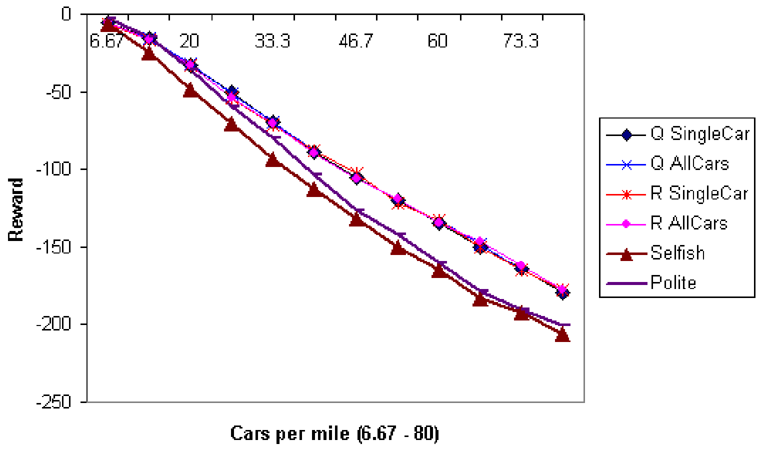


Figure 3: Average reward per car for the six methods as car density increases (3 lanes)

deceleration that occurs in a freeway environment. Most acceleration on the freeway does not involve the maximum acceleration possible by a car and deceleration often involves the driver removing his/her foot from the gas pedal.

Safety is always enforced in the traffic simulator. Although this is unrealistic, dealing with collisions and other related problems is beyond the scope of this paper. A safe distance is maintained between a car and the car in front of it. This safe distance is the distance the car would travel in one second based on the current speed. For example, if a car is going 70 mph, it will maintain a safe distance of 103 ft. between it and the car in front of it. If a car is going faster than the car in front of it, then there must also be enough room for it to slow down to the same speed as the car in front of it while still maintaining the safe distance.

3.2 Learning Performance

We investigated the four learning algorithms in both three and four lane roadways. Algorithms learned for 15,000 time steps with one car every 200 feet (396 cars for three lanes and 528 for four lanes). Figure 2 shows the learning curves for 15,000 time steps. Figure 2 shows the median run out of 10 separate runs.

As was seen in Moriarty and Langley (1998), the learning methods performed better than the heuristic methods and the polite method performed better than the selfish method. Surprisingly, all the learning methods performed similarly. The R-learning methods, which are tailored to this type of problem, generally performed worse than the Q-learning methods. The single car reward function performed better than the all cars reward function.

Learning stabilizes quickly for all the learning methods. All the learning methods have reached their approximate peak value by 3,000 time steps. There are two reasons for this. First, all the cars in the system are improving the value function. After 3,000 time steps, 3,000 * 396 = 1,584,000 updates have occurred. Second, the reward function is noisy and is only an approximation of the idealized single agent reward function.

We also tested learning performance of the these methods for the same number of time steps, but trained with 6 different sets of randomly placed cars for 2,500 time steps each. The results were similar to the continuous learning case, except all the learning methods performed better when trained on the 6 different sets (*** numbers).

3.3 Increasing Traffic Density

We also tested the robustness of the methods as the traffic density increased. Under normal traffic conditions (non-rush hour), traffic congestion is not much of a problem. Most drivers are able to maintain their desired speeds. However, as traffic density increases and the general flow of traffic begins to slow, methods that reduce this congestion become more important. Moriarty and Langley only tested traffic densities as high as 400 cars per 13.3 miles over 3 lanes. This is approximately one car every 500 feet. That is only 10 cars per mile, which is much less dense than rush hour conditions.

We trained the four learning methods on a 3 lane roadway with 396 cars for 30,000 time steps with 6 different random car configurations (each car configuration ran for 5,000 time steps). After learning, the value function is kept constant and is used to test the results as the traffic density increased. The methods were tested on traffic densities ranging from 100 cars to 1200 cars on a 3 lane roadway 5 miles long. The maximum density is approximately one car every 50 ft. This is much closer to rush hour traffic conditions than previous experiments. Simulating traffic densities higher than this was not useful because simulator parameters (i.e. maintaining safety) prevented cars from switching lanes at all. Each method was tested on 25 different random car configurations for each traffic density (100 – 1200) for 5,000 time steps each. This process was repeated three times, each time learning a new value function. Figure 3 shows the average results from these experiments.

As we saw in the learning curve experiments, all the learning methods perform similarly (even more so in this experiment). We hypothesize that because of the noisiness of the reward function, the methods are learning very similar value functions in the long run. Further experimentation is needed to confirm this.

As hypothesized, the learning methods perform better as the traffic density increases. For low densities, the performance of all the methods is similar. As the density increases, the learned methods gradually perform better than the heuristic methods. This is particularly remarkable given that the learning methods were trained with a fixed car density of 396 cars.

4 Conclusions and Future Work

To conclude, we give an overview of the basic procedure used for the car traffic control problem for translating a single agent problem to an approximating multi-agent problem. The first step is to specify these components for the single agent problem. Particular emphasis should be put on the reward function. Next, the states ***S***, the actions ***A***, and the reward function, ***R***, must be reformulated from the perspective of the smaller entities so as to still capture the basic information of the single agent problem. Section 4.2.2 provided a general method for doing this for any translation. Finally, a reinforcement learning method must be selected and applied to the problem. Each entity is treated as an independent agent, but all agents update the **same** value function. This setup can produce sub-optimal results with respect to the original single agent problem. Further research is required to bound this sub-optimality.

We examined car traffic control and showed that learning methods performed better than heuristic methods even without tuning learning parameters. Future work will target other domains, particularly domains where solutions using single agent are already known. This will provide a baseline to examine what the performance sacrifice is by reducing the problem to a multi-agent problem.

We attempted to model real traffic conditions and drivers in the simulator developed. However, the simulator used for experiments is still a long way from real life conditions. Experimentation in a more realistic environment would provide stronger motivation for implementing this type of system in real life.

Acknowledgements

Thanks to my machine learning classmates for their initial ideas and contributions and thanks to Charles Elkan for his guidance and analysis.

References

Ehlert, P. & Rothkrantz, L. (2001). A Reactive Driving Agent For Microscopic Traffic Simulation.

Mahadevan, S. (1996). Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results. *Machine Learning*, Special Issue on Reinforcement Learning (edited by Leslie Kaebling), vol. 22, pages 159-196.

Moriarty, D., & Langley, P. (1998). Distributed learning of lane-selection strategies for traffic management (Technical Report 98-2). Daimler-Benz Research & Technology Center, Palo Alto, CA.

Schwartz, A. (1993). A Reinforcement Learning Method for Maximizing Undiscounted Rewards. Proceedings of the Tenth Nation Conference on Machine Learning, pages 298-305.

Singh, S. (1994). Reinforcement Learning Algorithms for Average-Payoff Markovian Decision Processes. Proceedings of the Twelfth National Conference on Artificial Intelligence.

Sukthankar, R., Baluja, S., & Hancock, J. (1997). Evolving an intelligent vehicle for tactical reasoning in traffic. Proceedings of the IEEE International Conference on Robotics and Automation.

Sutton, R. S. & Barto, A. (1998). Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA.

Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England.