Multi-Agent Reinforcement Learning for Traffic Light Control

Marco Wiering MARCO@CS.UU.NL
University of Utrecht, Department of Computer Science, Postbox 80 089, 3508 TB Utrecht, The Netherlands

Abstract

This paper describes using multi-agent reinforcement learning (RL) algorithms for learning traffic light controllers to minimize the overall waiting time of cars in a city. The RL systems learn value functions estimating expected waiting times for cars given different settings of traffic lights. Selected settings of traffic lights result from combining the predicted waiting times of all cars involved. We investigate RL systems using different kinds of global communicated information between traffic light agents. We also show how the value functions can be used by the driving policies of cars to select optimal routes to destination addresses. The experimental results show that the RL algorithms can outperform non-adaptable traffic light controllers, and that optimizing driving policies is very useful.

1. Introduction

In traffic control, we are interested in enhancing social welfare for road-users such as minimizing traveling time or maximizing safety. An important traffic control problem is optimizing multiple cooperating traffic light controllers so that the total waiting time of cars before traffic lights is minimized. Since optimizing traffic light controllers by hand is a complex and tedious task, we study how reinforcement learning (RL) algorithms can be used for this goal.

Learning in multi-agent systems. The traffic light control problem consists of multiple traffic nodes (intersections) containing a set of traffic lights connected to each other by an infrastructure. When we model traffic lights and cars as agents, we want to optimally deal with the behavior of each agent and with their interactions. For optimizing the behavior of each agent in the multi-agent system (MAS) we can employ reinforcement learning (RL) algorithms such as Q-learning (Watkins, 1989). This is not completely new: Tan (1993) used multi-agent Q-learning (MAQ-L) for a predator-prey problem. Littman and Boyan (1993) used MAQ-L for network routing and showed that multi-agent RL provides useful tools for such complex problems. Mataric (1994) used RL to train a group of robots to collect pucks. Schaerf, Shoman, and Tennenholtz (1995) used a multi-agent RL system for learning load balancing. Crites and Barto (1996) used MAQ-L for training elevator dispatchers in a simulated environment and obtained better controllers than a set of fixed controllers used in practice.

Model-based RL for MAS. Most former approaches based on RL to optimize MASs used model-free (direct) RL such as Q-learning (Watkins, 1989) and TD(λ)-methods (Sutton, 1988) to optimize single agent behaviors. Previous comparison studies have shown advantages in using model-based RL (MBRL) (see Atkeson & Santamaria, 1997; Moore & Atkeson, 1993; Wiering, 1999). In model-based (indirect) RL, a transition model is estimated and dynamic programming-like methods are used to compute the value function mapping agent states to expected long term reward. This can speed up learning time dramatically. We will use MBRL to learn to estimate waiting times of cars given particular input states.

Co-learning. We also study how we can use the same RL systems and learned value functions to optimize the paths which cars take to arrive at their destination address. Since the value functions are then exploited by two types of agents (cars and traffic lights) and are adapted based on decisions of all agents, agents cooperatively learn the shared value functions. We will use the term *co-learning* for this joint learning strategy. Co-learning can be used if all agents share the goal of using policies which minimize the same value function.

Outline of this paper. Section 2 describes the traffic light simulator. Section 3 describes several MBRL methods employing different kinds of communication to optimize the traffic light controllers and also describes how they can be used for co-learning driving policies. Section 4 describes the experimental set-up and shows experimental results. Section 5 discusses the findings and Section 6 concludes.

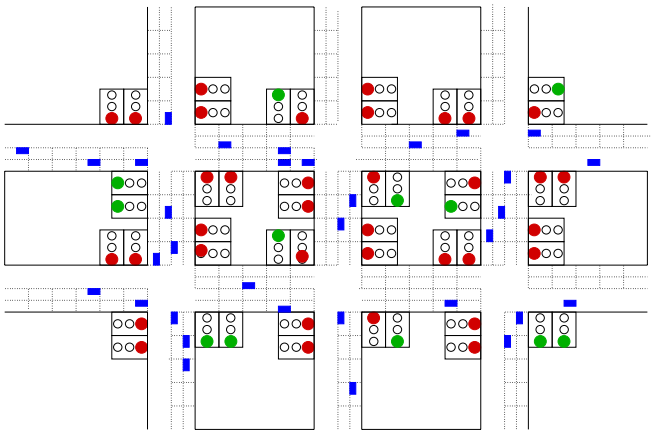

Figure 1. The traffic control problem. The goal is to learn traffic light policies which minimize the overall time cars need to go to their destination address. The complete network containing all 6 intersections is called the city. Entrees to and exits from the city are given by the open side-roads. At each intersection, there are 8 traffic lights operational. Before each traffic light there is a specific road-lane discretized into a number of possible places for cars.

2. Traffic Simulator

A traffic light control problem is given by a network where edges represent roads and nodes represent intersections where traffic lights are operational. There are cars coming from outside the system and these follow a policy to drive over the roads (while waiting for red traffic lights) until they have arrived at their destination address (see Figure 1).

Traffic light model. At each traffic node (intersection), there are 8 traffic lights operational: 4 for going straight ahead or to the right, and 4 for going to the left. We prevent accidents by not allowing traffic light states which make collisions between cars possible. Possible settings for traffic lights are: two traffic lights from opposing directions allow cars to go straight ahead or to turn right (2 possibilities), two traffic lights at the same direction of the intersection allow the cars from there to go straight ahead, turn right or turn left (4 possibilities). This makes a total of 6 possible decisions (actions) for each traffic node.[1]

Behavior of cars. Before each different traffic light, there is a specific road-lane discretized into a number of places. At each time-step, new cars are generated with a particular destination address and enter the network (city) at the last place of one of the side-entrance road-lanes. We assume for now that cars follow a random policy to arrive at the destination address along one of the shortest paths, although as we will see later, cars could also be adaptive and learn specific driving policies to minimize their traveling time. After new cars have been added, traffic light decisions are made and each car moves to the subsequent place if this is possible — if it is unoccupied or the car's predecessor is moved as well. Cars standing in front of a red traffic light are not moved.

All road-lanes have a limited capacity for storing cars (we used a maximum of 20 cars). Therefore, it may happen that cars standing before a green traffic light cannot continue, since the next road-lane is overcrowded (and its first car cannot drive). Our simulator is a discrete event simulator — we do not take acceleration/deceleration times into account nor do we model different car velocities. We are mostly interested in the co-operational behavior of traffic nodes and cars.

How to solve this problem? Traffic light control problems are often solved by optimizing the times each light is on green and red. These controllers are often made more advanced by using sensors to infer that at some point in time no traffic has arrived at a traffic light so that the light can be set to red. Although optimizing waiting times may work well, it remains a hard problem and decisions do not take the exact environmental state into account, possibly leading to sub-optimal controllers. That is why RL systems may be advantageous. These systems can learn to set a traffic light given a particular environmental input.

3. Reinforcement Learning for Traffic Light Control

In RL (Kaelbling, Littman & Moore, 1996) we try to optimize the behavior of an agent by letting it interact with the environment and learn from its obtained feedback (reward). An agent is situated in an environment, receives (virtual) sensory inputs and uses these to select an action by its policy. This policy is optimized by learning from the results of applying different action sequences given some input. A value function is used to estimate long term reward intake given that the agent observes a particular input (state) and selects actions according to its policy. Well-known algorithms for learning value functions are Q-learning (Watkins, 1989) and TD(λ)-learning (Sutton, 1988). These algorithms update the value function directly on the sequence of state/action pairs observed during the interaction of the agent with the environment. Model-based RL (Moore & Atkeson, 1993) first learns a transition model which estimates the probabilities of making state-transitions given particular actions and

[1] We have not allowed the possibility that cars from road-lanes of opposing directions turn left at the same time.

compute average rewards associated to these transitions. Then it uses dynamic programming-like methods (Bellman, 1961; Barto, Bradtke & Singh, 1995; Moore & Atkeson, 1993) to compute the value function. MBRL can significantly speed up learning (Atkeson & Santamaria, 1997; Moore & Atkeson, 1993; Wiering, 1999), although it requires a discrete representation of the input space and more storage space.

Estimating cumulative waiting time. Each cycle, cars can wait 1 time-step before a traffic light, or they can drive to the next place. Particular cars are standing in the queue (a row of cars standing before a traffic light without a gap between them), and their movement is dependent on their predecessors and traffic light setting. If cars are not standing in the queue, they always move one place further. If the light is green, the first car in the queue immediately crosses the intersection and starts on the last place of the road-lane for the next traffic light. After one or more time-steps, it ends up at some place in a new queue at the next traffic light or it escapes the city. The goal is to minimize the cumulative waiting time of all cars before all traffic lights met before exiting the city. To do this, each car learns to estimate its waiting time when the light is green or red and all car predictions are combined to make the decision of a traffic node.

Global description of the system. Each **car** is at a specific traffic-light (**tl** $\in$ [1..48]), a position in the queue (**place** $\in$ [1..20]), and has a particular destination address (**des** $\in$ [1..10]). Since there are 960 possible places in the network which may be occupied by a car, there are at least 2^{960} possible traffic situations. Since each car has a specific destination address, there are even more system states. A system state should be mapped to actions of all 6 traffic nodes, making a total of 6^6 possible global actions in our network. Thus, learning a central controller mapping system states to actions is infeasible. To deal with so many states and actions, we independently control traffic nodes — each controller receives as input the cars (traffic light, place, and destination)[2] standing for one of its 8 traffic lights and selects one of its 6 actions.

Making a traffic node decision. Suppose all cars would exactly know their waiting time until they arrive at the destination address given that their traffic light is currently set to red or green. Then, each car has a gain for having its light set to green. This gain equals the difference between its waiting time when the light is red and when it is green. If there are a number of cars standing before different traffic lights at a traffic node, we can choose the decision which maximizes the summed gains of all cars which profit from the decision. This decision would then be (locally) optimal.

Car-based value functions. For this goal, we will use car-based value functions stored in lookup tables. The car-based value functions estimate the total (discounted) expected waiting time before all traffic lights for each car until it arrives at the destination address given its current traffic light, place, and the decision of the light (red or green). We will write $Q([tl, place, des], action)$ to denote this value. We will write $V([tl, place, des])$ to denote the average waiting time (without knowing the traffic light decision) for a car at $(tl, place)$ until it has reached its destination address. Note that we use the destination address of the car, which may help to estimate its waiting time more accurately. The Q- and V-functions are distributively stored in the traffic light controllers which communicate with cars and other traffic lights. To learn the Q- and V-functions, traffic nodes may request V-values of other traffic nodes.

Given the current traffic situation, we make a choice for each traffic node *node* as follows. The gain-variable $W_{node}(A)$ computes the advantage (gain) of the decision of the traffic node, A, which sets two specific traffic lights to green:

Table 1. Selecting a traffic node decision by summing individual gains.

(1) For all A, one of the 6 decisions:
  (1.a) $W_{node}(A) = 0$
  (1.b) For all traffic lights tl set to green by A
    (1.b.1) For all cars in the queue at the traffic light tl, with their destination and place
      (1.b.1.a) $W_{node}(A) = W_{node}(A) + Q([tl, place, des], red) - Q([tl, place, des], green)$
(2) Select the decision A of the traffic node with maximal $W_{node}(A)$

If a car is not waiting in the queue (i.e. the car can still drive onwards until it meets the queue), we do not let it vote for the total waiting time, since the current decision of the traffic light does not affect the current transition of the car.

Although the Q-values are real numbers, we have used integers for representing the variables $W_{node}(A)$, which allows for exploring the results of decisions which seem almost as good as the currently best decision. Particular systems, however, suffered from this integer representation so that we had to use real numbers for their gain variable (we will later denote the use of real numbers by a system as system^r).

[2] We assume that in the near future it is possible for cars to communicate with intelligent traffic light controllers which enables them to send place and destination information in a real world application.

Communicating global information. Using MBRL we have different design choices for using current information about the state of other traffic lights for computing waiting times. This kind of global information could be communicated between traffic light agents. If we would know how many cars are standing at each next possible traffic light, we can improve the probabilistic estimates of the place where the first car will enter the next queue. Thus, we could predict waiting times more accurately by instantiating global (communicated) information. We will discuss three systems; the first, **TC-1** (Traffic Controller 1), does not use such communication and only uses local information for computing waiting times of cars, **TC-2** only uses global information for computing waiting times for the first car and local information for the other cars, and **TC-3** uses global information for computing waiting times for all cars.

The transition and reward functions. For computing the Q- and V-functions we use state transition probabilities and a reward function. The state transition function is given by a lookup table consisting of the following probabilities: $P([tl, place, des], L, [new\_tl, new\_place])$ where L denotes whether the light for tl is red or green. Note that the destination address des stays the same and that most car-states only have two possible transitions. Therefore the transition function does not contain a huge number of entries. If a car crosses a traffic node, we compute its state transition probability to the new traffic light, and new place as that place where it is (at a certain moment) for the first time standing in the new queue ($1 \leq new\_place \leq 20$). Finally, we compute probabilities $P(L|[tl, place, destination])$ which give the probability that the light is red or green for a car waiting at $(tl, place)$ with a particular destination. These probabilities are needed to compute the average waiting time $V([tl, place, destination])$. Finally, we use a reward function as follows: if a car stays at the same place, then $R([tl, place], [tl, place]) = 1$. Otherwise $R = 0$ (the car can advance).

TC-1: Computing the V- and Q-functions. For our first system, TC-1, we compute the Q-function as:

$$Q([tl, p, d], L) = \sum_{(tl', p')} P([tl, p, d], L, [tl', p'])$$
$$(R([tl, p], [tl', p']) + \gamma V([tl', p', d])) \qquad (1)$$

Where γ is the discount factor ($0 < \gamma < 1$) which ensures that Q-values are bounded. Thus, the expected discounted waiting time of a car given L equals the current waiting probability (which is 1 for a red light) plus the average waiting time from the next possible car-states.[3] We compute the V-function using the Q-function and the probabilities that the light is green or red for a car as follows:

$$V(]tl, p, d]) = \sum_{L} P(L|[tl, p, d])Q([tl, p, d], L)$$

After each simulation step we update the transition probabilities and compute the V-values and Q-values by Real Time Dynamic Programming (Barto, Bradtke & Singh, 1995) using a single value-function iteration.

TC-2. The first communicating system uses the number of cars standing at the next possible traffic lights (where the first car can go to) to compute the state-transition probabilities of the first car only. We estimate the transition probability of the car to a next traffic light tl' given that at tl' currently $K_{tl'}$, shortly written as K, cars are standing in the queue. This information should be communicated between traffic lights. To compute the value functions, we will make use of transition probabilities $P([tl, place, des], K, green, [tl', p])$. We only use this equation if the light is green:

$$Q([tl, p, d], green) = \sum_{(tl', p')} P([tl, p, d], K, green, [tl', p'])$$
$$(R([tl, p], [tl', p']) + \gamma V([tl', p', d]))$$

Note that here the value of p is always 1 — for the other cars we use Equation (1) to compute the Q-values. To compute V, we again take the average of the Q-values according to the probabilities that the light is green or red. Note that we do not use K as argument in the Q- and V-functions — K is not part of the car-state, but is communicated and then instantiated in the transition function to compute the value functions. This saves us from an additional dimension in the Q-function. The TC-2 system may work fine, but has particular shortcomings: (1) we only look ahead a single traffic light in the future — situations at traffic nodes some steps further are not used. (2) Predicting waiting times of other cars waiting in the queue does not immediately take advantage of this communication (although there is delayed, indirect, communication due to subsequent value-iteration steps).

TC-3. TC-3 uses global knowledge for computing waiting times for all cars. It uses state transition probabilities: $P([tl, place, des], L, K, [new\_tl, new\_place])$ for all cars which determine the state transition probabilities to the next traffic lights even while the car is not yet in a position to cross the intersection. The

[3] This equation very much resembles Bellman's equation: $Q(S, A) = \sum'_S P(S, A, S')(R(S, S') + \gamma V(S'))$.

V-function is computed by summing the expected discounted waiting time (EDWT) at the current traffic light and the EDWT from the next possible lights:

$$V([tl,p,d]) = W([tl,p,d]) + \sum_{L} P(L|[tl,p,d])$$
$$\sum_{(tl',p')} P([tl,p,d],K,L,[tl',p'])\gamma V([tl',p',d])$$

Where $W([tl,place,des])$ is the EDWT at the current traffic light, and K is the number of cars waiting at the next possible traffic light tl'. The W-function can be computed as:

$$W([tl,p,d]) = \sum_{L} P(L|[tl,p,d])Q'([tl,p,d],L)$$

where Q' (the intra-node Q-function) denotes the EDWT at the current light for cars given the decision of the traffic light:

$$Q'([tl,p,d],L) = \sum_{p'} P([tl,p,d],L,[tl,p'])$$
$$(R([tl,p],[tl,p']) + \gamma W([tl,p',d]))$$

The Q-values can finally be computed as follows:

$$Q([tl,p,d],L) = Q'([tl,p,d],L) +$$
$$\sum_{(tl',p')} P([tl,p,d],K,L,[tl',p'])\gamma V([tl',p',d])$$

We compute $P([tl,place,des],K,L,[tl',p])$ by tracking a car standing on a specific place. Thus, we record tuples $< tl,place,des,K,L,tl' >$ and finally associate them with p, the place where the car arrives in the next queue. If there are multiple states of a car on $(tl,place)$ with the same L and $K_{tl'}$, we only count the transition step to the next (tl',p) a single time (this is similar to the first visit sampling method (Singh & Sutton, 1996)).

Adapting the system parameters. For adapting the systems, we update the state transition probabilities after each time-step by tracking car-movements. Remember that the reward function is fixed (standing still costs 1, otherwise the reward/cost is 0). To compute transition probabilities, we just count the number of transitions from a car-state to all next car-states and divide these by the total number of transitions from that car-state.[4]

Co-learning driving policies. A nice feature of our car-based value functions is that they can be immediately used to select a path of traffic lights to the destination address. Note that our city (Figure 1) is like Manhattan and from one starting place to a destination address there can be multiple shortest paths. The non-adaptable systems generate at each traffic light what the options are to go from one traffic light to the next one in order to go to the destination address and select one of these randomly. Co-learning can be used to select among these shortest paths that path with minimal expected waiting time. For TC-1, we compare the values $V([tl',1,des])$ to determine the best next traffic light tl' for a car crossing an intersection. For TC-2 and TC-3, we can compute the Q-values for going to the next traffic light tl' using global information. We compute the values $Q(tl')$ for going to a next traffic light (given the current light tl) as follows:

$$Q(tl') = \sum_{p} P([tl,1,d],K,green,[tl',p])V([tl',p,d])$$

and choose the traffic light tl' with the lowest $Q(tl')$.

4. Experiments

We execute experiments with 10 systems: a random controller for each traffic node, a fixed controller which iterates over all traffic node decisions, a controller which lets the largest queues go first, a controller which tries to let most cars pass the intersection, and our three RL systems: TC-1, TC-2, and TC-3, with or without co-learning. For our experiments we use the city depicted in Figure 1.

Set-up of traffic simulations. The traffic pattern is a fully randomized pattern where random starting traffic lights at the border of the city (20 possibilities) are selected for each newly inserted car and a random destination addresses is used for the car (10 possibilities).[5] At each cycle (time-step), 1 to 8 cars are inserted in the city, all with different starting traffic lights, since cars cannot occupy the same initial place at the same traffic light. Therefore it is also possible that the traffic network becomes saturated, where cars are refused since we cannot add more cars when all 20 possible starting positions are occupied.

Systems and parameters. The Random system selects the decision at each traffic node randomly, the Fixed system starts with decision 1 for all traffic nodes for one time-step, then selects decision 2 at the next time-step, until it has selected all six decisions and starts again with decision 1. The Longest Q system counts the number of cars which would not have to wait for a red light for each decision and selects the traffic node decision leading to the maximum. The

[4]For particular systems, we have to take communicated state information into account as well.

[5]Due to particular impossible paths, generated cars cannot use all 200 combinations.

Table 2. Final waiting time results for different systems when adding 1-3 cars per time-step. Results are averages over 10 simulations. ^{r} = used a real number for the gain variable.

System	1 car	2 cars	3 cars
Random	10.9 ± 0.4	19.7 ± 1.2	174 ± 11
Fixed	5.6 ± 0.05	9.5 ± 0.4	69 ± 6
Longest Q	0.47 ± 0.02	1.50 ± 0.04	4.4 ± 0.2
Most Cars	0.47 ± 0.02	1.60 ± 0.07	4.6 ± 0.4
TC-1^r	0.47 ± 0.02	1.50 ± 0.03	3.9 ± 0.3
TC-1^r co	0.45 ± 0.03	1.44 ± 0.07	3.9 ± 0.4
TC-2^r	0.47 ± 0.02	1.52 ± 0.06	4.2 ± 0.2
TC-2^r co	0.45 ± 0.02	1.36 ± 0.06	3.9 ± 0.3
TC-3	0.46 ± 0.02	1.48 ± 0.07	4.0 ± 0.3
TC-3 co	0.44 ± 0.02	1.36 ± 0.05	3.6 ± 0.3

Table 3. Final waiting time results and the nr. of refused cars (1K =1000) for the systems when adding 4 cars per time-step. * = 20% randomness is used in the action selection.

System	waiting time	refused cars
Random	171 ± 13	26K ± 1K
Fixed	70 ± 7	226K ± 11K
Longest Q*	3683 ± 1297	941K ± 43K
Most Cars	706 ± 1830	6K ± 9K
TC-1	190 ± 108	15K ± 1K
TC-1 co	70 ± 22	3K ± 2K
TC-2	128 ± 45	10K ± 2K
TC-2 co	58 ± 22	3K ± 2K
TC-3*	106 ± 12	49K ± 24K
TC-3 co*	89 ± 20	14K ± 17K

Most cars system examines how many cars can pass an intersection given some traffic node decision, and selects the decision which is expected to let most cars (0-2) cross an intersection. TC-1, TC-2, and TC-3, with or without co-learning use $\gamma = 0.99$, one value-function iteration per time-step and no exploration, except for systems which get stuck in dead networks where no cars can drive anymore (which sometimes happens with the TC-3, Longest Q, and Most cars systems), for which we add 20% random actions to the decision policy. We let each system run until 50,000 cars have exited the city and record simulation results after each 2000 cars have left the city. Results are averages over 10 simulations.

Experimental results. Table 2 shows the final (after 50,000 steps) average waiting time results for the last 2000 cars exiting the city when adding 1 to 3 cars per time-step. When adding a single car at each cycle, TC-3 with co-learning works best, closely followed by the other co-learning RL systems. The random system performs worst with a waiting time which is more than 23 times longer than the best algorithms. When adding 2 cars the results are quite similar. TC-3 and TC-2 with co-learning works best followed by TC-1 with co-learning. When adding 3 cars, TC-3 with co-learning works best followed by the other RL systems. Longest Q performs 22% worse and Most cars performs 28% worse than TC-3 with co-learning. The random and fixed systems again come last and even result in saturating behavior — see Figure 2(A). Note that although the differences are not so large, TC-3 with co-learning always significantly (t-test, $p_{chance} < 0.01$) outperforms all fixed systems.

Adding four cars. Table 3 shows the results when adding 4 cars. Here the network starts to saturate for all algorithms. Therefore not only the average waiting time is important, but also the number of refused cars. For Longest Q and TC-3, we had to add 20% noise in the action selection, since otherwise they got stuck in traffic situations where no cars could move anymore and no cars were able to enter the city. Such "dead" network states result from deterministic policies which set lights to green for cars which cannot cross the intersection, since the next road-lane is full and the next (or the one after the next) traffic light is set to red.

TC-2 with co-learning works best, followed by TC-1 with co-learning. They have the lowest waiting times and refuse the lowest number of cars. Note that the number of refused cars would make aligning traffic networks more crowded. Apparently, optimizing driving policies in busy traffic situations is very useful here. TC-3 refuses many cars during the initial learning phase, but finally obtains the best performance of the non co-learning systems. The Most cars algorithm results in fluctuating performance (waiting times). It does not refuse so many cars, though, which is different from the Longest Q system which refuses by far the most cars.

Saturation behavior for adding more cars. Figure 2(A) shows the total number of refused cars during a run when we increase traffic loads and Figure 2(B) shows the average final waiting times. When adding 5-8 cars, TC-2 with co-learning refuses the least number of cars. It is followed by TC-1 with co-learning. The Longest Q system performs worst. The fixed system also refuses many cars and this explains why its average waiting time is shortest for highly crowded traffic. The random system works quite well for very crowded roads; it seems that for such cases random decisions work reasonably well. The Most cars algorithm performs quite well, but suffers from fluctuating performance levels. All systems can use co-learning of driving policies to minimize the number of refused cars. The reason that TC-2 with co-learning works

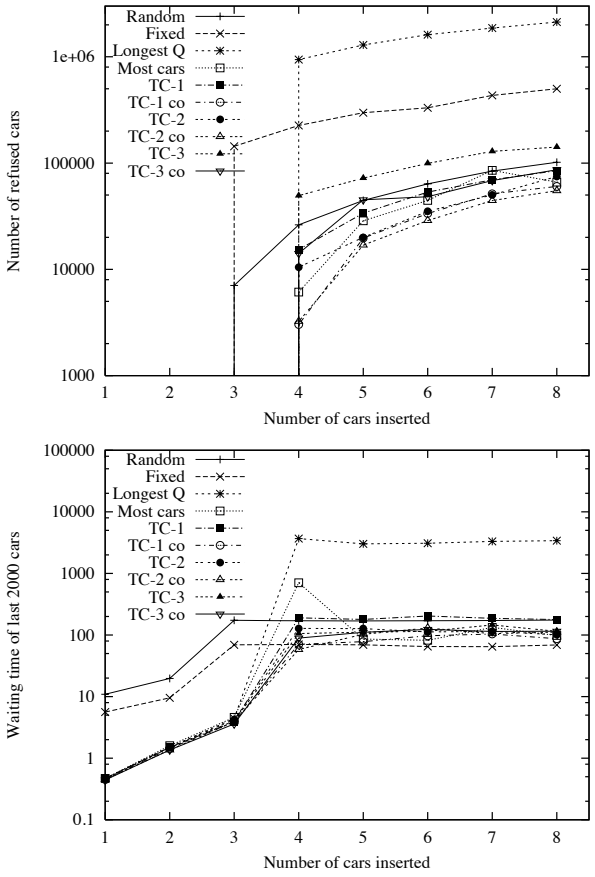

Figure 2. A comparison between the different adaptable and fixed systems on more or less crowded traffic patterns. (A): The average number of refused cars during an entire run. (B): The average waiting time of the last 2000 cars exiting the city. Results are averages over 10 simulations.

best, may be that it continuously adapts its policy, thereby making it non-stationary. Therefore it can react when particular decisions do not make sense, like setting the light to green while the first car cannot go to an overcrowded next road-lane. TC-3 is not able to continuously change its policy, and therefore it sometimes ends up in dead networks (when used without randomness). The reason is that only intra-Q values are adapted when cars remain waiting, and this is sometimes not sufficient to change the outcome of the voting process, since inter-Q values may have a large impact on the decision. Furthermore, communicating K is less useful if K is almost always 20 and the first car cannot drive. The Longest Q system suffers a lot from deadlock situations (no car can drive given some decisions of the traffic lights), but it is strange that even with 20% randomness it cannot overcome its problems (with more randomness it performs better, but even with 90% randomness, it performs worse than the random system).

5. Discussion

For low traffic loads, constructing good (near-optimal) fixed controllers is not difficult, since all traffic nodes can operate locally. Therefore the gain in using RL for learning traffic light controllers is quite small, although learning driving policies is still useful. When we increase traffic load, the amount of interaction between traffic nodes increases, and the locally well performing fixed systems do not work well anymore. Furthermore, the dynamics of crowded traffic patterns are complex so that it is hard to design better controllers. Here, using RL systems for traffic light control is clearly beneficial. Co-learning driving policies is also very useful, since it helps to direct traffic flow in the city.

Co-learning. Learning driving policies at the same time as learning traffic light controllers show interesting co-learning phenomena: traffic nodes which are quite busy and thus have a hard task minimizing overall waiting time are relieved by the intelligent driving policies circumventing such intersections. Thereby the cars are reactively spreading in the city and help to minimize the shared value functions.

Communication. The use of communicated information can help the RL systems to optimize traffic light controllers. Since traffic nodes are highly interdependent when regulating highly crowded traffic, we could also design different communication schemes in which traffic node decisions are communicated. We are currently studying methods for efficiently evaluating global decisions in this way.

Related work. Thorpe and Anderson (1996) used direct RL to learn traffic controllers on a simulated traffic control problem consisting of a network of 4 $\times$ 4 traffic light controllers. They modelled average speed, queueing and acceleration/ deceleration of cars. The controller was trained on a single intersection after which it was copied to the other intersections. Results showed that using their best state representation (which indicates which segments of the roads were occupied by cars) RL learned to outperform algorithms which used fixed waiting times or allowed the largest queue to go first. A big difference between their and our approach is that their traffic node policy selects decisions based on a combined representation of the local traffic situation. To deal with the explosive number of states, they abstract away from a lot of information. Instead, we use car-based value functions and a voting scheme for selecting actions. This has the advantage that (local) optimal controllers may be obtained if the value functions are accurate, while we still do not suffer from huge state spaces. Furthermore, the car-based value functions can be used by the driving policies.

Moriarty and Langley (1998) also used RL for distributed traffic control. Their approach enabled cars to learn lane selection strategies from experience with a traffic simulator. Experimental studies showed that learned strategies let drivers more closely match their desired speeds than hand-crafted controllers and reduce the number of lane changes. Their approach also focuses on distributed car-based controllers, which makes it easy to take specific desires/goals of drivers into account such as desired speed or destination.

6. Conclusion

We have presented a set of multi-agent model-based RL systems for traffic light control which can also be used for optimizing driving policies for cars. Experimental results show that the RL systems can outperform a number of non-adaptable systems. One of the systems, TC-3, uses global communication between traffic lights and is able to surpass the performance of the other algorithms when the traffic is not very crowded. If the networks start to saturate when we increase traffic load, the RL systems clearly outperform fixed controllers and also profit a lot from co-learning driving policies.

In future work, we would like to test our systems on more realistic traffic simulators in which we also want to add public transport which should get priorities for crossing roads, since they carry more passengers. In another direction, we want to examine whether other multi-agent problems can profit from simple communication between agents. For this, we want to use MBRL algorithms since they are quickly able to deal with different kinds of instantiated information. The problems we want to focus on are network routing, (process) scheduling, robot soccer, and forest fire control.

Acknowledgements

Thanks to Prof. F.C.A. Groen, Ben Kröse, Stephan ten Hage and the anonymous reviewers for many helpful comments.

References

Atkeson, C. G., & Santamaria, J. C. (1997). A comparison of direct and model-based reinforcement learning. In *Proceedings of the International Conference on Robotics and Automation.*

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 72*, 81–138.

Bellman, R. (1961). *Adaptive control processes.* Princeton University Press.

Crites, R., & Barto, A. (1996). Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems 8* (pp. 1017–1023). Cambridge MA: MIT Press.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Littman, M., & Boyan, J. (1993). A distributed reinforcement learning scheme for network routing. *Proceedings of the First International Workshop on Applications of Neural Networks to Telecommunication* (pp. 45–51). Hillsdale, New Jersey.

Mataric, M. J. (1994). *Interaction and intelligent behavior.* Doctoral dissertation, AI Laboratory, Massachusetts Institute of Technology, Cambridge, MA.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning, 13*, 103–130.

Moriarty, D., & Langley, P. (1998). Learning cooperative lane selection strategies for highways. *Proceedings of the Fifteenth National Conference on Artificial Intelligence.* Menlo Park, CA: AAAI Press.

Schaerf, A., Shoman, Y., & Tennenholtz, M. (1995). Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research, 2*, 475–500.

Singh, S. P., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning, 22*, 123–158.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning, 3*, 9–44.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. *Proceedings of the Tenth International Conference on Machine Learning* (pp. 330–337).

Thorpe, T., & Anderson, C. (1996). *Traffic light control using SARSA with three state representations.* IBM Corporation, Boulder.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* Doctoral dissertation, King's College, Cambridge.

Wiering, M. A. (1999). *Explorations in efficient reinforcement learning.* Doctoral dissertation, Intelligent Autonomous Systems Group, University of Amsterdam.