Reinforcement Learning of Traffic Light Controllers Adapting to Traffic Congestion.

Conference Paper · January 2005

Source: DBLP

CITATIONS
51

READS
1,369

5 authors, including:

Emil Nijhuis
Universitätsspital Basel
9 PUBLICATIONS **184** CITATIONS
SEE PROFILE

Bram Bakker
Amsterdam University Medical Center
36 PUBLICATIONS **1,706** CITATIONS
SEE PROFILE

REINFORCEMENT LEARNING OF TRAFFIC LIGHT CONTROLLERS ADAPTING TO TRAFFIC CONGESTION

Merlijn Steingröver Roelant Schouten Stefan Peelen Emil Nijhuis
Bram Bakker

Intelligent Autonomous Systems group, Informatics Institute
University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam

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. In this paper we describe the optimization of traffic light controllers using a multi-agent, model-based reinforcement learning or approximate real-time dynamic programming approach. Our methods optimize individual traffic lights locally, but the optimization takes into account traffic congestion at neighboring traffic lights, such that there is implicit cooperation between them. We show, using experiments performed with a traffic simulator, that this approach outperforms existing methods.

1 Introduction

Traffic is an integral and important part of modern society. However, with an increasing population and increasing mobility, traffic jams are becoming a more and more common sight, especially in and around large urban areas. This leads to significant economical damage and environmental pollution.

Traffic in urban areas is, in large part, controlled by means of traffic lights. Inefficient configuration of traffic lights can lead to unnecessarily long waiting times for cars and even to traffic jams. Such inefficient configuration is unfortunately still the case in a lot of urban areas; many traffic lights are based on a "fixed cycle" protocol, which basically means that the lights are set to green for a fixed amount of time and consecutively to red for a fixed amount of time. Usually this is not optimal, since such a policy does not take into account the current traffic situation at a particular traffic light, meaning that situations can occur in which a long queue of waiting cars is being held up by a shorter queue (or in the worst case, by no queue at all). What one would want to happen in such a situation is that the policy decides to let longer queues have an "advantage" over the shorter ones in order to improve the total traffic flow; but this should not lead to situation where cars at a quieter part of a junction will never be able to continue their journey.

The question that arises from these considerations is how to determine a policy that optimally takes into account the particular traffic situation at each traffic light junction. "Hand-coding" all possible situations around a traffic junction and determining the traffic light configurations according to those situations is infeasible due to the large number of possible traffic situations and the complexity of the control problem.

A promising approach is to make use of machine learning techniques. Such methods allow the control system to automatically learn a good, or even optimal policy. Techniques that have been used for this problem include genetic algorithms [5] and fuzzy logic [6]. This paper uses reinforcement learning (RL) to optimize the traffic light controllers in a traffic network. We use a model-based RL algorithm, alternatively viewed as approximate real-time dynamic programming. The methods we propose optimize an individual traffic light junction in the traffic network locally, but the optimization takes into account traffic congestion at neighboring junctions in the traffic network, such that there is implicit cooperation between the traffic lights.

The next section describes RL and its application to traffic light optimization. In section 3 we describe our new methods. Section 4 presents experiments performed with this approach. Section 5, finally, presents conclusions and future work.

2 Reinforcement learning and traffic control

2.1 Reinforcement Learning

RL has been applied to various domains with considerable success. One of the most impressive results so far is in backgammon [7], where a program based on this method is able to play at world-class expert level.

RL is a technique for learning control strategies for autonomous agents from trial and error [4, 3]. The agents interact with the environment by trying out actions, and use resulting feedback (reward and the consecutive state) to reinforce behavior that leads to desired outcomes.

RL usually formalizes the learning problem as a Markov Decision Process (MDP). An MDP consists of a finite set of states $S = s_1, s_2, ...$, a finite set of actions $A = a_1, a_2, ...$ that an agent can perform, state transition probabilities $P(s, a, s')$ which are the probabilities that an agent in state s will arrive in state s' after performing action a, and finally the expected real-valued reward $R(s, a, s')$ which defines the expected immediate scalar reward which the agent will receive when it executes action a in state s and arrives in state s'.

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

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

where R_t is the reward at time t, and γ is a factor which discounts rewards further in the future.

Most RL algorithms (such as Q-learning [9, 4]) define and learn a *value function* $Q(s, a) : S$ x $A \rightarrow \mathbb{R}$, whose values directly represent the returns attainable from different states for different actions. Once the optimal value function Q^* has been found, the optimal policy for the agent simply corresponds to: $\pi^*(s) = \arg\max_a Q^*(s, a)$.

2.2 The traffic simulator

There are two common approaches of modeling traffic: macroscopic models, which model large scale traffic densities, and microscopic models, which model individual cars and their interactions. In this paper we deal with detailed traffic patterns and traffic light control in a (simulated) city environment, hence we use a microscopic model. We use a freely available traffic simulator, the *Green Light District* (GLD) traffic simulator [11, 12].[1]

The main entities in the simulations are cars driving through a traffic network and traffic light controllers controlling individual traffic lights at traffic light junctions (see figure 1). The cars drive on roads consisting of different driving lanes going in different directions at a junction. Every lane consists of a discrete number of positions, each of which can be occupied by a car. As in th real world, particular driving lanes and corresponding traffic lights constrain the direction in which a car can go. For example, the left lane may be for cars that wish to go straight on or turn left, the right lane may be for cars that wish to go turn right.

The network is initialized without any cars. At the edges of the network there are nodes which let cars enter the network; these "edge nodes" have a so-called "spawning rate" that defines at what rate it will let new cars enter the network; the edge nodes are, at the same time, possible destination nodes for the cars. When a car is spawn at an edge node, it is randomly assigned another edge node as its destination. It then makes use of a simple path planning method which calculates the shortest path from its current position to its destination node, so it knows at each junction which way it has to go. When there are multiple shortest paths, one is chosen randomly.

A car drives toward its destination at a fixed speed, obeying simple rules such as stopping when a car in front of it stops, and stopping for red lights. A car automatically chooses the correct driving lane, for example choosing the right lane when at the next junction it must turn right. When a car

[1] Available from `http://sourceforge.net/projects/stoplicht`.

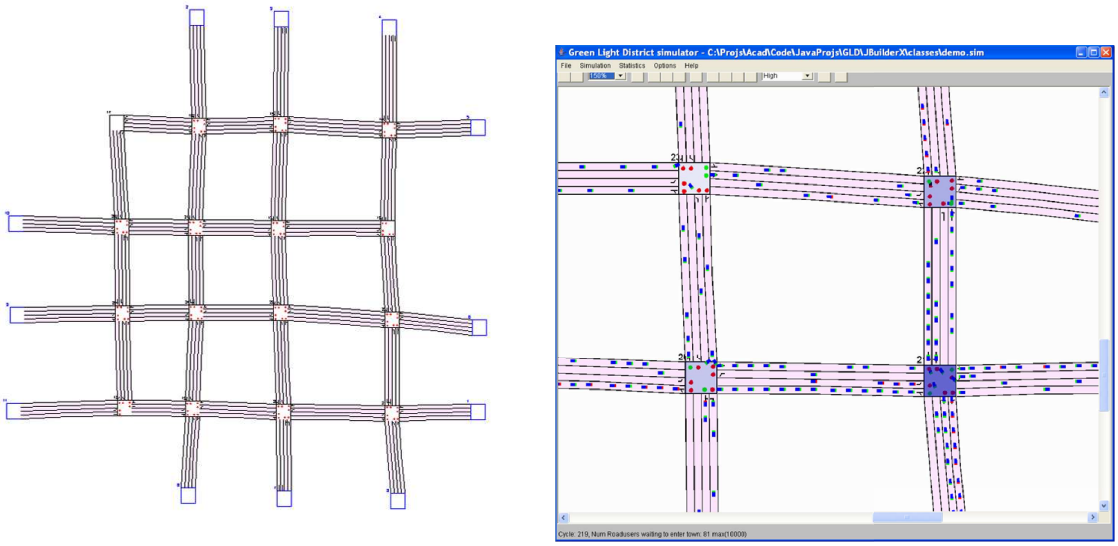

Figure 1: Fig. a (left). The GLD traffic network used in this paper, with 12 edge nodes, 15 traffic sign junctions and 1 junction that does not have a traffic light. Fig. b (right). Snapshot of traffic in part of the traffic network in one run. Darker nodes indicate more congested traffic.

arrives at its destination edge, it disappears from the simulation but the amount of time it had to wait during its trip is recorded.

2.3 Traffic light control with RL

The goal in our traffic light control problem is to minimize the total travel time of all cars. The reward function is, therefore, based on waiting time of cars.

Two different types of state representations are obvious candidates for this domain:

- One is a **traffic light-based** state representation, which represents all possible traffic configurations around a traffic light junction [8]. Such an approach requires each traffic light junction to learn a value function that maps all possible traffic configurations to total waiting times of all cars. This quickly leads to a very large statespace, because there are many possible configurations of cars on the driving lanes.

- The other is a **car-based** state representation, which represents the world state from the perspective of individual cars. In this approach, value functions estimate waiting times 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. Note that cars do not have to represent the value functions themselves—this can be done by the traffic light—but the representation is car-based.

Because of the problem of large state spaces associated with the traffic light-based state representation, the methods we use employ the car-based approach [11, 12, 10].

In the simplest case [11, 12], the state s of a car is described as a quadruple $s = [n, u, p, d]$, where n is the car's current traffic junction; u is the driving direction, associated with one specific traffic light at this junction; p is the position in the queue in front of that traffic light, and d is the final destination of the car.[2]

Each individual traffic light i at a traffic light junction n can be set to either *red* or *green*. At each junction only feasible, "legal" traffic light configurations are considered which ensure that cars

[2]Leaving d, the final destination of the car, out of the state representation does not appear to affect performance of these types of controllers very much [11], and is more realistic with respect to current real-world traffic light controllers. Future work is based on this more realistic assumption.

from different lanes will not collide. Thus, lanes that cross will never get a green light at the same time. The set of legal traffic light configurations constitutes the action set A at the junction.

The estimated optimal traffic light configuration at junction n, A_n^{opt}, is computed as follows:

$$A_n^{opt} = \max_{A_n} \sum_i \sum_{s \in L} (Q(s, green) - Q(s, red)) \tag{2}$$

where L is the set of states of all cars currently waiting in front of traffic light i at this junction n, and $Q(s, green)$ and $Q(s, red)$ represent the expected waiting time until the destination is reached when the associated traffic light is set to green or to red, respectively. Thus, the traffic light controller sums over all cars and traffic lights at a junction, and chooses that traffic light configuration which maximizes the combined gain of all cars at the junction.

To estimate the Q-values, a version of real-time dynamic programming is used [2]. The reward function is deterministic and straightforward:

$$R(s, a, s') = \begin{cases} -1 & s = s' \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Thus, if and only if a car stays in the same place the reward (cost) is -1. If a row of cars must wait for a red traffic light, all of them have this cost, which is why longer queues in front of traffic lights will in general get preference over shorter ones. Note that the reward function is independent of the total travel time: only taking longer than is strictly necessary is punished. State transition probabilities $P(s, a, s')$ are estimated online using a standard maximum likelihood modeling method: state transitions from s to s' given a are counted, and divided by the total number of state transitions from s.

After each iteration, a single real-time dynamic programming update is done for all occupied states:

$$Q(s, a) \leftarrow \sum_{s'} P(s, a, s')(R(s, a, s') + \gamma V(s')) \tag{4}$$

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

In sum, the multi-agent system learns a single model consisting of $P(s, a, s')$ and $R(s, a, s')$, and a single value function $Q(s, a)$ estimating waiting time for individual cars. Decisions optimizing combined utility of all agents are made by combining Q-values of cars waiting at a traffic light junction.

3 RL adapting to congestion

3.1 Coordination and congestion

The RL method described above has already been shown to outperform a variety of alternative, non-learning, handcoded traffic light controllers [10, 11, 12]. Nevertheless, there is still considerable room for improvement. In the basic method(called Traffic Controller-1 or TC-1 [10, 11, 12]), traffic light configurations are selected for each junction individually. Some form of coordination between traffic lights at different junctions in the network may improve overall performance. The explicit coordination of traffic lights along a route by programming "green waves" is one well-known example.

Secondly, and related to the issue of coordination, in the method described above traffic light decisions are made based on local state information around a single junction. Taking into account traffic situations at other junctions may improve overall performance, especially if these traffic situations are highly dynamic and may include both free flowing traffic and traffic jams. For example, there is no use in setting lights to green for cars that, further on in their route, will have to wait anyway because there traffic is congested completely.

The two methods proposed in this paper extend the method described above based on these considerations. The basic idea in both new methods is to stick to the idea of local optimization of traffic light junctions, in order to keep the methods computationally feasible. However, traffic light junctions will now take into account, as extra information, the amount of traffic at neighboring junctions, a "congestion factor". This means that next to local state information, more global (but not completely global) information is also used for traffic light optimization, and there is a form of implicit cooperation between neighboring traffic light junctions.

3.2 TC-SBC method

The first method proposed in this paper takes into account traffic situations at other places in the traffic network by expanding the state space with another dimension. This extra dimenstion represents the amount of traffic congestion at neighboring traffic junctions. We call this method the "State Bit for Congestion" (TC-SBC) method. Now the state s is described as a quintuple $s = [n, u, p, d, b]$, where b is a bit indicating for a car whether its destination lane at the next traffic light junction is congested or not. First, for each car a real-valued congestion factor c is computed:

$$c = \frac{w_j}{D_j} \tag{5}$$

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. b is computed according to

$$b = \begin{cases} 1 & c > \theta \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where θ is a parameter acting as a threshold.

Like before, the model consisting of $P(s, a, s')$ and $R(s, a, s')$ and the value function $Q(s, a)$ are learned online using maximum likelihood estimation and real-time dynamic programming. But unlike before, now the system effectively learns different state transition probabilities and value functions when the neighboring roads are congested and when they are not congested. This makes sense, because the state transition probabilities and expected waiting times are likely to be widely different for these two cases. This allows the system to effectively learn different controllers for the cases of congestion and no congestion. An example of how this may lead to improved traffic flow is that a traffic light may learn that if (and only if) traffic at the next junction is congested, the expected waiting time for cars in that direction is almost identical when this traffic light is set to green compared to when it is set to red. Therefore, it will give precedence to cars going in another direction, where there is no congestion. At the same time, this will give the neighboring congested traffic light junction more time to resolve the congestion.

A somewhat similar idea was proposed by Wiering [10] and investigated in another, simpler traffic simulator. Those methods (called TC-2 and TC-3) can be understood as partially expanding the state set for certain state transition probabilities $P(s, a, s')$ such that the number of cars at the destination lane of the next traffic junction is taken into account. However, the actual state used in the value function $Q(s, a)$ is not changed from what it was in TC-1, meaning that an unusual adaptation of the real-time dynamic programming update must be used and the $Q(s, a)$ values may not (and usually will not) converge, but instead keep oscillating between different values. Results for TC-2 and TC-3 were mixed but indicate that sometimes they can improve performance.

3.3 TC-GAC method

A disadvantage of the TC-SBC method described in the previous section is that it increases the size of the state space. That was, in fact, the main reason for restricting the state expansion to only one bit indicating congestion for only the immediate neighbor. The second new method investigated in this paper does not expand the state space, but instead uses the congestion factor c (described above in eq. 5) in a different way.

Rather than quantizing the real-valued c to obtain an additional state bit, it is used in the computation of the estimated optimal traffic light configuration:

$$A_n^{opt} = \max_{A_n} \sum_i \sum_{s \in L} (1 - c(s))(Q(s, green) - Q(s, red)). \tag{7}$$

where $c(s)$ is the appropriate congestion factor for each car in L. Thus, 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).

We call this method the "Gain Adapted by Congestion" (TC-GAC) method. The advantage of this method is that it does not increase the size of the state space, while making full use of real-valued congestion information. The disadvantage is that unlike TC-SBC, TC-GAC never learns

anything permanent about congestion, and the method is specific to this particular application domain and not easily generalizable. Note that GAC can in fact be combined with SBC, because it is, in a sense, an orthogonal extension. We call the combination of the two the TC-SBC+GAC method.

4 Experiments and Results

4.1 Test domains

We tested our algorithms using the GLD simulator (section 2.2). We used the same traffic network as the one used in [11, 12] (see figure 1). Since the original experiments already showed that RL clearly outperforms a variety of handcoded controllers [11, 12], in the experiments described below we do not compare with handcoded controllers, but only with the original RL algorithm (TC-1, section 2.3). We did not use the test results as described in the original papers, but ran all tests ourselves, to make sure both approaches were tested using exactly the same conditions.

The original experiments [11, 12, 10] were all done with fixed spawning rates. We replicate some of those experiments, but also added experiments (using the same road network) with *dynamic* spawning rates. In these experiments, spawning rates change over time, simulating the more realistic situation of changes in the amount of traffic entering and leaving a city, due to rush hour etc.

In all simulations we measure the Average Trip Waiting Time (ATWT), which corresponds to the amount of time spent waiting rather than driving. In all simulations we measure the Average Trip Waiting Time (ATWT), which corresponds to the amount of time spent waiting rather than driving. It is possible that traffic becomes completely jammed due to the large numbers of cars entering the road network. In that case we set ATWT to a very large value (50). Because of the complexity of the simulations, it is impossible to exactly compare with globally optimal solutions, but the lowest reported ATWTs appear to very good traffic flow.

We did preliminary experiments to find a good parameter setting of the congestion threshold parameter θ for the TC-SBC method. $\theta = 0.8$ gave the best results in all cases, also for the TC-SBC+GAC method, so we are using that value in the experiments described below. It is worth noting that even though TC-SBC adds another parameter, apparently one parameter value works well for different variations of the problem and the algorithm, and no parameter tuning is required for every specific case.

4.2 Experiment 1: fixed spawning rate

As described above, we performed simulations to compare our new congestion-based RL controllers (TC-SBC, TC-GAC, TC-SBC+GAC, section 3) to the original RL controller (TC-1, section 2.3) in one of the original test problems. This test problem corresponds to Wiering's experiment 1 [11, 12] and uses the road network depicted in figure 1. The spawning rate is fixed and set to 0.4. Each run corresponds to 50,000 cycles. For each condition 5 runs were done and the results were averaged.

Figure 2a shows the overall results of the experiment. The Average Trip Waiting Time (ATWT) is depicted over time. Each curve represents the average of 5 runs. The most important result is that all of our new methods lead to better performance, i.e. lower ATWT, than the original method TC-1, especially in the long run. The best performance in this test problem is obtained by TC-SBC+GAC, the method that combines both ways of using congestion information. Interestingly, in the initial stages of learning this method has, for a while, the worst average performance, but then it apparently learns a very good value function and corresponding policy, and reduces ATWT to a very low level.

Figure 1b shows a snapshot of traffic during one run, where the controller was TC-SBC+GAC. A darker traffic junction indicates a more congested traffic situation around that junction. The lower right junction is fairly congested, which results in the traffic lights leading to that junction to be set to red more often and for longer periods of time.

4.3 Experiment 2: dynamic spawning rate

Experiment 2 uses the same traffic network (figure 1) as experiment 1, but uses dynamic spawning rates rather than fixed ones. These simulations similarly last for 50,000 cycles and start with

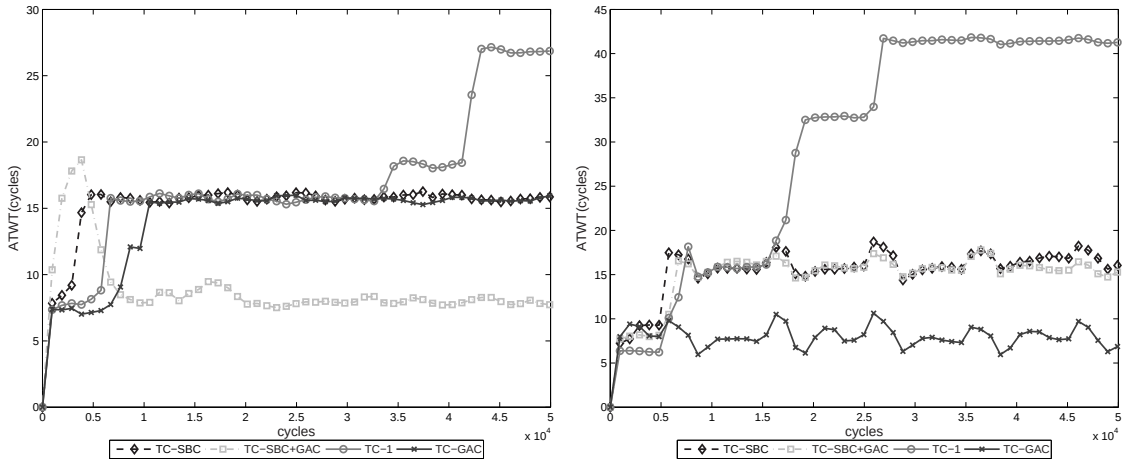

Figure 2: Fig. a (left). Performance of our congestion-based controllers versus TC-1, in the fixed spawning rate simulation (experiment 1). Fig. b (right). Performance of our congestion-based controllers versus TC-1, in the dynamic spawning rate simulation (experiment 2, "rush hours").

spawning rate 0.4. Every run is divided into five blocks of 10,000 cycles, within each of which at cycle 5,000 "rush hour" starts and the spawning rate is set to 0.7. At cycle 5,500 the spawning rate is set to 0.2, and at cycle 8,000 the spawning rate is set back to 0.4 again. The idea is that especially in these dynamic situations traffic flow may benefit from our new methods, which dynamically take into account congestion.

Figure 2b shows the results. As in experiment 1, all of our new methods outperform the original, TC-1 method. As expected, the difference is actually more pronounced (note the scale of the graph): TC-SBC, TC-GAC, and TC-SBC+GAC are much better in dealing with the changes in traffic flow due to rush hours. Rush hours do lead to longer waiting times temporarily (the upward "bumps" in the graph). However, unlike the original, TC-1 method, our methods avoid getting complete traffic jams, and after rush hours they manage to reduce the Average Trip Waiting Time. The best method in this experiment is TC-GAC.

5 Conclusions and future work

From the experiments it can be concluded that the addition of implicit traffic light cooperation, by letting traffic lights take into account the level of traffic congestion at neighboring traffic lights, has a beneficial influence on the performance of the algorithms. The algorithms using this new approach always perform better than the original method in the experiments. A general lesson that may be drawn from this is that fruitful cooperation between agents can in certain cases be accomplished by letting individual, locally optimizing agents take into account (for example through the state representation) limited or compressed information concerning the state of other, perhaps neighboring agents.

Possible future work includes investigating the possibilities of even more global cooperation between traffic light junctions. This can be accomplished by letting a car take into account not only the next traffic light, but also the ones that lie further ahead on its path. An enhancement like this applied to TC-SBC would require the state space to become very large in case of large traffic networks, which may lead to computational problems. For TC-GAC, on the other hand, it is not obvious how traffic congestions measured at more than one point should be incorporated in the adaptive gain computation (eq. 7).

A second avenue for future research deals with the problem of rapidly increasing state spaces mentioned above. Hierarchical methods may be used that use a divide-and-conquer strategy to deal with large problems. States are clustered into higher-level states, resulting in smaller state spaces and easier optimization problems [1].

In a third line of future work we aim to simulate accidents. One idea is that accidents immedi-

ately lead to local congestions, which could be dealt with by traffic light controllers using methods similar to the ones investigated here. Adaptive traffic light controllers may be able to adaptively route traffic around accident spots.

Of course, the ultimate goal is that these or similar RL methods will be tested and implemented in real traffic systems. Important steps before that goal can be reached are more results concerning robustness of the controllers, and ways of realizing the required sensor capabilities (through loops in the road, cameras, and/or communication with cars).

Acknowledgments

The research reported here is part of the Interactive Collaborative Information Systems (ICIS) project, supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024.

References

[1] B. Bakker, Z. Zivkovic, and B. Kröse. Hierarchical dynamic programming for robot path planning. In *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[2] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

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

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

[5] H. Taale, T. Bäck, M. Preuss, A. E. Eiben, J. M. de Graaf, and C. A. Schippers. Optimizing traffic light controllers by means of evolutionary algorithms. In *Proceedings of EUFIT'98*, 1998.

[6] K. K. Tan, M. Khalid, and R. Yusof. Intelligent traffic lights control by fuzzy logic. *Malaysian Journal of Computer Science*, 9-2, 1995.

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

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

[9] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.

[10] M. Wiering. Multi-agent reinforcement learning for traffic light control. In *Proceedings of the Seventeenth International Conference (ICML'2000)*, pages 1151–1158, 2000.

[11] 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.

[12] 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.