Intelligent Traffic Light Control

Marco Wiering

Jelle van Veenen

Jilles Vreeken

Arne Koopman

institute of information and computing sciences, utrecht university

technical report UU-CS-2004-029

www.cs.uu.nl

Intelligent Traffic Light Control

Marco Wiering, Jelle van Veenen, Jilles Vreeken, and Arne Koopman
Intelligent Systems Group
Institute of Information and Computing Sciences
Utrecht University
Padualaan 14, 3508TB Utrecht, The Netherlands
email: marco@cs.uu.nl

July 9, 2004

Abstract

Vehicular travel is increasing throughout the world, particularly in large urban areas. Therefore the need arises for simulating and optimizing traffic control algorithms to better accommodate this increasing demand. In this paper we study the simulation and optimization of traffic light controllers in a city and present an adaptive optimization algorithm based on reinforcement learning. We have implemented a traffic light simulator, Green Light District, that allows us to experiment with different infrastructures and to compare different traffic light controllers. Experimental results indicate that our adaptive traffic light controllers outperform other fixed controllers on all studied infrastructures.

Keywords: Intelligent Traffic Light Control, Reinforcement Learning, Multi-Agent Systems (MAS), Smart Infrastructures, Transportation Research

1 Introduction

Transportation research has the goal to optimize transportation flow of people and goods. As the number of road users constantly increases, and resources provided by current infrastructures are limited, intelligent control of traffic will become a very important issue in the future. However, some limitations to the usage of intelligent traffic control exist. Avoiding traffic jams for example is thought to be beneficial to both environment and economy, but improved traffic-flow may also lead to an increase in demand [Levinson, 2003].

There are several models for traffic simulation. In our research we focus on microscopic models that model the behavior of individual vehicles, and thereby can simulate dynamics of groups of vehicles. Research has shown that such models yield realistic behavior [Nagel and Schreckenberg, 1992, Wahle and Schreckenberg, 2001].

Cars in urban traffic can experience long travel times due to inefficient traffic light control. Optimal control of traffic lights using sophisticated sensors and intelligent optimization algorithms might therefore be very beneficial. Optimization of traffic light switching increases road capacity and traffic flow, and can prevent traffic congestions. Traffic light control is a complex optimization problem and several intelligent algorithms, such as fuzzy logic, evolutionary algorithms, and reinforcement learning (RL) have already been used in attempts

to solve it. In this paper we describe a model-based, multi-agent reinforcement learning algorithm for controlling traffic lights.

In our approach, reinforcement learning [Sutton and Barto, 1998, Kaelbling et al., 1996] with road-user-based value functions [Wiering, 2000] is used to determine optimal decisions for each traffic light. The decision is based on a cumulative vote of all road users standing for a traffic junction, where each car votes using its estimated advantage (or gain) of setting its light to green. The gain-value is the difference between the total time it expects to wait during the rest of its trip if the light for which it is currently standing is red, and if it is green. The waiting time until cars arrive at their destination is estimated by monitoring cars flowing through the infrastructure and using reinforcement learning (RL) algorithms.

We compare the performance of our model-based RL method to that of other controllers using the *Green Light District simulator* (GLD). GLD is a traffic simulator that allows us to design arbitrary infrastructures and traffic patterns, monitor traffic flow statistics such as average waiting times, and test different traffic light controllers. The experimental results show that in crowded traffic, the RL controllers outperform all other tested non-adaptive controllers. We also test the use of the learned average waiting times for choosing routes of cars through the city (co-learning), and show that by using co-learning road users can avoid bottlenecks.

This paper is organized as follows. Section 2 describes how traffic can be modelled, predicted, and controlled. In section 3 reinforcement learning is explained and some of its applications are shown. Section 4 surveys several previous approaches to traffic light control, and introduces our new algorithm. Section 5 describes the simulator we used for our experiments, and in section 6 our experiments and their results are given. We conclude in section 7.

2 Modelling and Controlling Traffic

In this section, we focus on the use of information technology in transportation. A lot of ground can be gained in this area, and Intelligent Transportation Systems (ITS) gained interest of several governments and commercial companies [Ten-T expert group on ITS, 2002, White Paper, 2001, EPA98, 1998].

ITS research includes in-car safety systems, simulating effects of infrastructural changes, route planning, optimization of transport, and smart infrastructures. Its main goals are: improving safety, minimizing travel time, and increasing the capacity of infrastructures. Such improvements are beneficial to health, economy, and the environment, and this shows in the allocated budget for ITS.

In this paper we are mainly interested in the optimization of traffic flow, thus effectively minimizing average traveling (or waiting) times for cars. A common tool for analyzing traffic is the traffic simulator. In this section we will first describe two techniques commonly used to model traffic. We will then describe how models can be used to obtain real-time traffic information or predict traffic conditions. Afterwards we describe how information can be communicated as a means of controlling traffic, and what the effect of this communication on traffic conditions will be. Finally, we describe research in which all cars are controlled using computers.

2.1 Modelling Traffic

Traffic dynamics bare resemblance with, for example, the dynamics of fluids and those of sand in a pipe. Different approaches to modelling traffic flow can be used to explain phenomena specific to traffic, like the spontaneous formation of traffic jams. There are two common approaches for modelling traffic; macroscopic and microscopic models.

2.1.1 Macroscopic models.

Macroscopic traffic models are based on gas-kinetic models and use equations relating traffic density to velocity [Lighthill and Whitham, 1955, Helbing et al., 2002]. These equations can be extended with terms for build-up and relaxation of pressure to account for phenomena like stop-and-go traffic and spontaneous congestions [Helbing et al., 2002, Jin and Zhang, 2003, Broucke and Varaiya, 1996]. Although macroscopic models can be tuned to simulate certain driver behaviors, they do not offer a direct, flexible, way of modelling and optimizing them, making them less suited for our research.

2.1.2 Microscopic models.

In contrast to macroscopic models, microscopic traffic models offer a way of simulating various driver behaviors. A microscopic model consists of an infrastructure that is occupied by a set of vehicles. Each vehicle interacts with its environment according to its own rules. Depending on these rules, different kinds of behavior emerge when groups of vehicles interact.

Cellular Automata. One specific way of designing and simulating (simple) driving rules of cars on an infrastructure, is by using cellular automata (CA). CA use discrete partially connected cells that can be in a specific state. For example, a road-cell can contain a car or is empty. Local transition rules determine the dynamics of the system and even simple rules can lead to chaotic dynamics. Nagel and Schreckenberg (1992) describe a CA model for traffic simulation. At each discrete time-step, vehicles increase their speed by a certain amount until they reach their maximum velocity. In case of a slower moving vehicle ahead, the speed will be decreased to avoid collision. Some randomness is introduced by adding for each vehicle a small chance of slowing down. Experiments showed realistic behavior of this CA model on a single road with emerging behaviors like the formation of start-stop waves when traffic density increases.

Cognitive Multi-Agent Systems. A more advanced approach to traffic simulation and optimization is the Cognitive Multi-Agent System approach (CMAS), in which agents interact and communicate with each other and the infrastructure. A cognitive agent is an entity that autonomously tries to reach some goal state using minimal effort. It receives information from the environment using its sensors, believes certain things about its environment, and uses these beliefs and inputs to select an action. Because each agent is a single entity, it can optimize (e.g., by using learning capabilities) its way of selecting actions. Furthermore, using heterogeneous multi-agent systems, different agents can have different sensors, goals, behaviors, and learning capabilities, thus allowing us to experiment with a very wide range of (microscopic) traffic models.

Dia (2002) used a CMAS based on a study of real drivers to model the drivers' response to travel information. In a survey taken at a congested corridor, factors influencing the choice of route and departure time were studied. The results were used to model a driver population, where drivers respond to presented travel information differently. Using this population,

the effect of different information systems on the area where the survey was taken could be simulated. The research seems promising, though no results were presented.

2.2 Predicting Traffic

The ability to predict traffic conditions is important for optimal control. For example, if we would know that some road will become congested after some time under current conditions, this information could be transmitted to road users that can circumvent this road, thereby allowing the whole system to relieve from congestion. Furthermore, if we can accurately predict the consequences of different driving strategies, an optimal (or at least optimal for the predicted interval) decision can be made by comparing the predicted results.

The simplest form of traffic prediction at a junction is by measuring traffic over a certain time, and assuming that conditions will be the same for the next period. One approach to predicting is presented in [Ledoux, 1996], where neural networks are used to perform long-term prediction of the queue length at a traffic light. A multi-layer perceptron [Rumelhart et al., 1986] is trained to predict the queue length for the next time-step, and long-term predictions can be made by iterating the one-step predictor. The resulting network is quite accurate when predicting ten steps ahead, but has not yet been integrated into a controller.

A traffic prediction model that has been applied to a real-life situation, is described in [Wahle and Schreckenberg, 2001]. The model is a multi-agent system (MAS) where driving agents occupy a simulated infrastructure similar to a real one. Each agent has two layers of control; one for the (simple) driving decision, and one for tactical decisions like route choice. The real world situation was modelled by using detection devices already installed. From these devices, information about the number of cars entering and leaving a stretch of road are obtained. Using this information, the number of vehicles that take a certain turn at each junction can be inferred. By instantiating this information in a faster than real-time simulator, predictions on actual traffic can be made. A system installed in Duisburg uses information from the existing traffic control center and produces real-time information on the Internet. Another system was installed on the freeway system of North Rhine-Westphalia, using data from about 2.500 inductive loops to predict traffic on 6000 km of roads.

2.3 Controlling traffic by communicating traffic conditions

Once accurate (predictive) information is available, there are several ways of communicating it to road users. Drivers could be presented with information through dynamic road signs, radio, or even on-board navigation systems. Several studies have shown the effects of the availability of relevant information.

Levinson (2003) uses a micro-economic model to consider the cost of a trip, and increases system reliability, since congestions can be better avoided. This results in a supply curve shift. Experiments show that increasing the percentage of informed drivers reduces the average travel time for both informed and uninformed drivers. The travel time reduction is largest in crowded traffic. In the case of unexpected congestions (for example due to accidents) informed travellers reduce their travel time by switching routes, but as a result of this the alternative routes become more crowded, possibly increasing travel times for uninformed drivers.

Emmerink et al. (1996) present the results of a survey taken amongst drivers in Amsterdam. The results show that 70% of the drivers sometimes use information presented

through radio or variable message signs to adapt their routes. Both media are used in similar ways, and commuters are less likely to be influenced by information than people with other trip purposes. Business drivers indicated that they would be willing to pay for in-vehicle information.

Simulation could be used to test control strategies before they are implemented in real-life environments. Ben-Akiva et al. (2003) tested different strategies for a large highway project, such as highway access control, drivers' route choice, and lane control. The simulator offered a way of testing different configurations of vehicle-detectors, and showed that interacting control systems could actually worsen traffic conditions. Integration of strategies requires careful analysis of the impact of each component to avoid interference. A simulator shows to be a useful tool for such an analysis.

2.4 Vehicle Control

It is a well-known fact that traffic flow would increase drastically if all drivers would drive at the same (maximum) speed. Another fact is that this will never happen if you let drivers decide. In this section we first show how vehicles could learn to cooperate. We then describe an ambitious research program that aims to control all vehicles by on-board computers.

Moriarty and Langley (1998) have used reinforcement learning 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, like ours, 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.

In the California Partners for Advanced Transit and Highways (PATH) program, the Automated Highway System (PATH-AHS) project aims to completely automate traffic [Horowitz and Varaiya, 2000]. Cars on special roads would travel in so-called platoons. A platoon is a number of cars that travel at high speed, with little distance in between. Each car controls its own speed and lateral movement, and makes sure it follows the leader. The leader navigates the platoon, and makes sure that there is enough space between platoons. In order to optimize flow, a platoon leader receives information about the optimal speed from a roadside coordinating system. Because of this, and the fact that there is little distance in between cars in a platoon, an AHS is said to be able to increase road capacity by a factor of about four.

Another aspect of traffic control is controlling traffic lights in a way that minimizes the time drivers have to wait. We will describe previous research in this area and our car-based, multi-agent reinforcement learning algorithm in section 4. First we will discuss reinforcement learning.

3 Reinforcement Learning

Reinforcement learning [Sutton, 1988, Watkins, 1989, Kaelbling et al., 1996] is used to learn agent control by letting the agent (for example a car) interact with its environment and learn from the obtained feedback (reward signals). Using a trial-and-error process, a reinforcement learning (RL) agent is able to learn a policy (or plan) that optimizes the cumulative reward

intake of the agent over time. Reinforcement learning has been applied successfully in particular stationary environments such as in games: e.g. in backgammon [Tesauro, 1992] and chess [Baxter et al., 1997], in maze-like environments, e.g. in [Thrun, 1992, Moore and Atkeson, 1993, Wiering, 1999], and robot control tasks. Reinforcement learning has also been applied to find good solutions for difficult multi-agent problems such as elevator control [Crites and Barto, 1996], network routing [Littman and Boyan, 1993], and traffic light control [Wiering, 2000].

In this section, we will first describe Markov Decision Problems (MDPs) as a model of the task and environment. Dynamic programming [Bellman, 1957] can be used to compute an optimal action selection policy if the transition and reward functions of the MDP are known a-priori. Reinforcement learning can be used to learn to optimize a policy if the environment is a-priori unknown. We will explain dynamic programming and reinforcement learning and end with a discussion about some research on multi-agent reinforcement learning.

3.1 Markov Decision Problems

Markov decision problems can be used for modelling the interaction of an agent with its environment. A MDP $M =< S, A, P, R >$, in which we consider a finite set of environmental states $S = \{S_1, S_2, S_3, \ldots, S_n\}$, a finite set of actions A of the agent, and discrete time steps $t = 1, 2, 3, \ldots$. Let s_t denote the state at time t, and $a_t = \pi(s_t)$ the action, where π represents the agent's policy mapping states to actions. The transition function P with elements $P_{ij}(a) := p(s_{t+1} = j | s_t = i, a_t = a)$ for $i, j \in S$ defines the transition probability to the next state s_{t+1} given s_t and a_t. A reward function R maps state/action/state tupels $(i, a, j) \in S \times A \times S$ to scalar reinforcement signals $R(i, a, j) \in \mathbb{R}$. A discount factor $\gamma \in [0, 1]$ discounts later against immediate rewards.

3.2 Dynamic Programming

The agent's goal is to select actions that maximize the expected long-term cumulative discounted reinforcement, given an arbitrary (initial) state $s_0 \in S$. The value $V^\pi(s)$ is a prediction of the expected discounted cumulative reward to be received in the future, given that the agent is currently in state s and policy π will be used in the future:

$$V^\pi(i) = E(\sum_{k=0}^{\infty} \gamma^k R(s_k, \pi(s_k), s_{k+1}) | s_0 = i)$$

The expectancy operator E is used to average over all possible future paths through state/action space.

Action evaluation functions (Q-functions) $Q^\pi(i, a)$ return the expected future discounted reward for selecting action a in state i, and subsequently executing policy π:

$$Q^\pi(i, a) = \sum_j P_{ij}(a)(R(i, a, j) + \gamma V^\pi(j))$$

where V^π is defined as:

$$V^\pi(i) = \max_a Q^\pi(i, a)$$

By setting

$$\pi(i) = \arg\max_a Q^\pi(i, a)$$

for all states i, we then iteratively improve the policy, since we make an effective look-ahead step by recomputing the value functions V and Q and the policy π. The definition:

$$Q^*(i,a) = \sum_j P_{ij}(a)(R(i,a,j) + \gamma V^*(j))$$

is known as the Bellman optimality equation [Bellman, 1957]. Here Q^* and V^* are the optimal value functions. Basically the Bellman optimality equation says that looking one step ahead does not change the Q-function if it is already optimal. Furthermore, it has been proven that there is only one optimal Q^* and V^* function. Therefore, by iteration, we can find the optimal value functions. The well known dynamic programming algorithm called value iteration [Bellman, 1957] repeats the previous 3 equations for all states and actions, until the value function does not change anymore. For practical purposes, the iteration is often stopped once the value function changes hardly anymore (the largest update of a V-value is smaller than some ϵ). The value iteration algorithm is given below:

1. Initialise the Q-values and V-values (e.g. to 0)
2. Repeat (3-5) while termination criterion not met
3. Make an "update" for the Q-values:
$$Q(i,a) := \sum_j P(i,a,j)(R(i,a,j) + \gamma V(j))$$
4. Compute the new value function:
$$V(i) := \max_a Q(i,a)$$
5. Adapt the policy so that in each state the action with maximal current Q-value is selected:
$$\pi(i) := \arg\max_a Q(i,a)$$

3.3 Reinforcement Learning

In reinforcement learning the transition P and reward functions R are a-priori unknown. Therefore, the agent has to learn a policy by interacting with the environment that provides feedback about the goodness of particular state-action pairs. Reinforcement learning algorithms usually learn to update the Q-function after each performed action. The basic cycle of a reinforcement learning agent is:

1. Perceive the current state: s_t
2. Use the policy to select an action: $a_t = \pi(s_t)$
3. Receive a reward r_t and examine the new state s_{t+1} after the action was performed
4. Update the Q-function using the information (s_t, a_t, r_t, s_{t+1})
5. Set $t := t + 1$

Usually, the Q-function is initialized to all zero-values, and therefore the initial policy is completely random. By exploring the results of actions, and learning the best (current) policy from these, the reinforcement learning agent stochastically improves its behavior.

3.3.1 Q-learning.

Several reinforcement learning algorithms exist. The most commonly used algorithm is Q-learning [Watkins, 1989, Watkins and Dayan, 1992]. Q-learning does not use a model of the environment and updates the Q-function after each step using the following equation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

Where $0 < \alpha \leq 1$ is the learning rate. If the learning rate is decreased after each time-step in a specific way, and all state-action pairs are tried infinitely often, the use of Q-learning converges to the optimal Q-function and policy for Markov decision problems.

3.3.2 Model-based reinforcement learning.

In reinforcement learning we initially do not posess a model containing transition and the reward functions, but we can learn these from the observations received during the interaction with the world. Inducing a model from experiences can simply be done by counting the frequency of observed experiences. To this end our agent uses the following variables:

$C_{ij}(a)$:= number of transitions from state i to j after executing action a.
$C_i(a)$:= number of times the agent has executed action a in state i.
$R_{ij}(a)$:= sum over all immediate rewards received after executing action a in state i and stepping to state j.

A maximum likelihood model (MLM) is computed as follows:

$$\hat{P}_{ij}(a) := \frac{C_{ij}(a)}{C_i(a)}, \text{ and, } \hat{R}(i, a, j) := \frac{R_{ij}(a)}{C_{ij}(a)} \tag{1}$$

After each experience the variables are adjusted and the MLM is updated. In deterministic environments one experience per state/action pair (SAP) is sufficient to infer the true underlying model. In stochastic environments, however, we need to visit each state-action pair many times to get reasonable estimates. This also means that we stochastically improve the policy.

Dynamic programming (DP) techniques could immediately be applied to the estimated model, but online DP tends to be computationally very expensive. To speed up DP algorithms, efficient update-step management is often performed. This can be done by real-time dynamic programming [Barto et al., 1995], which uses only few updates of the value functions using value-iteration steps at each time-step, thereby only partially propagating updates back in time. Another efficient algorithm for recomputing the value functions is prioritized sweeping [Moore and Atkeson, 1993] which manages the updates in a way so that the largest updates are performed first.

3.4 Multi-agent Reinforcement Learning

In multi-agent systems, each agent tries to optimize its own behavior and may be able to communicate with other agents to make global coordination possible. Since optimizing the behavior of each agent in a MAS requires huge engineering efforts, we want to employ reinforcement learning or other optimization algorithms for automatically optimizing agent

behavior. 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. 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. Brauer and Weiss (1997) used RL to learn to schedule jobs to resources. Wiering, Salustowicz, and Schmidhuber (1999) used RL to learn soccer strategies from scratch. Nowadays, it is clear that RL provides useful, although not well understood, algorithms for optimizing MAS and we expect that the number of applications will grow continuously. Stone and Veloso (1997) and Wiering et al. (1999) are good surveys of multi-agent learning systems, but the field is rapidly expanding.

4 Traffic Light Control

Traffic light optimization is a complex problem. Even for single junctions there might be no obvious optimal solution. With multiple junctions, the problem becomes even more complex, as the state of one light influences the flow of traffic towards many other lights. Another complication is the fact that flow of traffic constantly changes, depending on the time of day, the day of the week, and the time of year. Roadwork and accidents further influence complexity and performance.

In practice most traffic lights are controlled by fixed-cycle controllers. A cycle of configurations is defined in which all traffic gets a green light at some point. The split time determines for how long the lights should stay in each state. Busy roads can get preference by adjusting the split time. The cycle time is the duration of a complete cycle. In crowded traffic, longer cycles lead to better performance. The offset of a cycle defines the starting time of a cycle relative to other traffic lights. Offset can be adjusted to let several lights cooperate, and for example create green waves.

Fixed controllers have to be adapted to the specific situation to perform well. Often a table of time-specific settings is used to enable a light to adapt to recurring events like rush hour traffic. Setting the control parameters for fixed controllers is a lot of work, and controllers have to be updated regularly due to changes in traffic situation. Unique events cannot be handled well, since they require a lot of manual changes to the system. Fixed controllers could respond to arriving traffic by starting a cycle only when traffic is present, but such vehicle actuated controllers still require lots of fine-tuning.

Most research in traffic light control focuses on adapting the duration or the order of the control cycle. In our approach we do not use cycles, but let the decision depend on the actual traffic situation around a junction, which can lead to much more accurate control. Of course, our approach requests that information about the actual traffic situation can be obtained by using different sensors or communication systems. We will first describe related work on intelligent traffic light control, and then describe our car-based reinforcement learning algorithm.

4.1 Related Work in Intelligent Traffic Light Control

4.1.1 Expert Systems.

An expert system uses a set of given rules to decide upon the next action. In traffic light control, such an action can change some of the control parameters. Findler and Stapp (1992)

describe a network of roads connected by traffic light-based expert systems. The expert systems can communicate to allow for synchronization. Performance on the network depends on the rules that are used. For each traffic light controller, the set of rules can be optimized by analyzing how often each rule fires, and the success it has. The system could even learn new rules. Findler and Stapp showed that their system could improve performance, but they had to make some simplifying assumptions to avoid too much computation.

4.1.2 Prediction-based optimization.

Tavladakis and Voulgaris (1999) describe a traffic light controller using a simple predictor. Measurements taken during the current cycle are used to test several possible settings for the next cycle, and the setting resulting in the least amount of queued vehicles is executed. The system seems highly adaptive, and maybe even too much so. Since it only uses data of one cycle, it could not handle strong fluctuations in traffic flow well. In this case, the system would adapt too quickly, resulting in poor performance.

Liu et al. (2002) introduce a way to overcome problems with fluctuations. Traffic detectors at both sides of a junction and vehicle identification are used to measure delay of vehicles at a junction. This is projected to an estimated average delay time using a filter function to smooth out random fluctuations. The control system tries to minimize not only the total delay, but the summed deviations from the average delay as well. Since it is no longer beneficial to let a vehicle wait for a long time, even if letting it pass would increase the total waiting time, this introduces a kind of fairness. Data of about 15 minutes is used to determine the optimal settings for the next cycle, and even using a simple optimization algorithm, the system performs well compared to preset and actuated controllers.

4.1.3 Fuzzy Logic.

Tan et al. (1995) describe a fuzzy logic controller for a single junction that should mimic human intelligence. Fuzzy logic offers a formal way of handling terms like "more", "less", "longer" etc., so rules like "if there is more traffic from north to south, the lights should stay green longer" can be reasoned with. The fuzzy logic controller determines the time that the traffic light should stay in a certain state, before switching to the next state. The order of states is predetermined, but the controller can skip a state if there is no traffic in a certain direction. The amount of arriving and waiting vehicles are quantized into fuzzy variables, like many, medium and none. The activation of the variables in a certain situation is given by a membership function, so when there are 5 cars in the queue, this might result in an activation of 25% of 'many' and 75% of 'medium'. Fuzzy rules are used to determine if the duration of the current state should be extended. In experiments the fuzzy logic controller showed to be more flexible than fixed controllers and vehicle actuated controllers, allowing traffic to flow more smoothly, and reducing waiting time. A disadvantage of the controller seems to be its dependence on the preset quantification values for the fuzzy variables. They might cause the system to fail if the total amount of traffic varies. Furthermore, the system was only tested on a single junction.

Lee et al. (1995) studied the use of fuzzy logic in controlling multiple junctions. Controllers received extra information about vehicles at the previous and next junctions, and were able to promote green waves. The system outperformed a fixed controller, and was at its best in either light or heavy traffic. The controller could easily handle changes in traffic flow, but

required different parameter settings for each junction.

Choi et al. (2002) also use fuzzy logic controllers, and adapted them to cope with congested traffic flow. Comparisons with fixed fuzzy-logic traffic light controllers indicated that this enhancement can lead to larger traffic flow under very crowded traffic conditions.

4.1.4 Evolutionary Algorithms.

Taale et al. (1998) compare using evolutionary algorithms (a (μ, λ) evolution strategy [Rechenberg, 1989]) to evolve a traffic light controller for a single simulated intersection to using the common traffic light controller in the Netherlands (the RWS C-controller). They found comparable results for both systems. Unfortunately they did not try their system on multiple coupled intersections, since dynamics of such networks of traffic nodes are much more complex and learning or creating controllers for them could show additional interesting behaviors and research questions.

4.1.5 Reinforcement Learning.

Reinforcement learning for traffic light control has first been studied by Thorpe [Thorpe, 1997, Thorpe and Andersson, 1996], but Thorpe's approach is different from ours. He used a traffic light-based value function, and we used a car-based one. Thorpe used a neural network for the traffic-light based value function which predicts the waiting time for all cars standing at the junction. This means that Thorpe's traffic light controller have to deal with a huge number of states, where learning time and variance may be quite large. Furthermore, Thorpe used a somewhat other form of RL, SARSA (State-Action, Reward-State Action) with eligibility traces [Sutton, 1996], and we use model-based RL.

Thorpe trained only a single traffic light controller, and tested it by instantiating it on a grid of 4×4 traffic lights. The controller can decide to let either traffic on the north-south axis or on the east-west axis pass. A neural network is used to predict the Q-values for each decision, based on the number of waiting cars and the time since the lights last changed. The goal state is the state in which there are no cars waiting.

The system outperformed both fixed and rule-based controllers in a realistic simulation with varying speed. Performance is said to be near optimal. Controllers trained on a single junction had the same performance as controllers trained in a network.

4.1.6 Intelligent Agents.

Roozemond (1998) describes an intelligent agent architecture for traffic light control. Intelligent traffic signalling agents (ITSAs) and Road Segment Agents (RSAs) try to perform their own tasks, and try to achieve local optimality. One or more Authority Agents can communicate with groups of ITSAs and RSAs for global performance. All agents act upon beliefs, desires, and capabilities. No results were presented.

4.2 Our Approach

We are interested in the behavior of cars and traffic lights. Each cycle, cars can wait 1 timestep at a traffic light, they can drive to the next position of a road-lane, or they can cross an intersection and go to another road-lane. We define a (waiting) queue as all cars that are immediately affected by the setting of their traffic light, because they will have to brake for a

red light. Note that a car standing on the first place of a road-lane is always in the queue. If a car is the first in the queue and the light is green, it crosses the intersection and starts on the last place (cell) of the new road-lane for the next traffic light. After one or more time-steps, it will finally end up at some place in a new queue at the next traffic light or it reaches its destination (and exits the infrastructure).

The goal is to minimize the cumulative waiting time of all cars before all traffic lights met before exiting the city. There are two ways of approaching this problem:

- **Traffic-light based controllers.** We can make a controller for each traffic node, taking into account environmental inputs such as the number of cars waiting at each of the 8 directions and learning a value function mapping environmental states and traffic node decisions to the overall waiting time until all cars standing at the intersection have exited the city.
- **Car-based controllers.** We can make a predictor for each car to estimate the waiting time of the car alone when the light is green or red and combine all car predictions to make the decision of the traffic light.

Making a control policy and learning a value function for each traffic node (as done by Thorpe) has the disadvantage that the number of situations for a traffic node can be huge. Furthermore, it is difficult to compute total trip waiting times for all road users standing at the traffic node, since this quantity has a huge variance. Although we could cope with these issues by designing a special traffic-node representation summing individual waiting times of cars, this system does not allow for communicating information to cars or for making decisions for cars (e.g. which paths they should take).

4.2.1 Car-based traffic light control.

Therefore, we chose the second possibility that allows for a lot of flexibility, since we have the option to model each car's destination, position, and possibly speed. This is also a natural system, since if cars would exactly know their overall waiting time until their destination (note that this would require static traffic patterns) when the light is green or red, a voting system that adds all waiting times for different traffic node decisions can be used to minimize the overall waiting time. Furthermore, the number of states for a car is not so large. For example if we use the information that the car is at a traffic node, occupies some place, is at some direction, and has some destination, this makes a feasible number of car-states which can be stored in lookup tables. Note that we take the destination address of cars into account. This allows us to compute the global waiting time until the car has reached its destination. Finally, by computing different expected waiting times for a car when it is standing at one of the different directions (traffic lights) before a traffic node, we can also use this information to choose a path to the destination address for each car. This makes co-adaptivity (co-learning) possible in which both the traffic nodes and the driving policies of cars are optimized.

4.2.2 Car-based value functions.

To optimize the settings of traffic lights, we can simply sum all expected waiting times of all cars given all possible choices of a traffic light, and select the decision which minimizes the overall waiting time at the intersection. For this, we need to learn a value function which

estimates for each car how long its waiting time will be until it has reached its destination address given that the light is green or red.

4.2.3 Global design of the system.

Each **car** is at a specific traffic-node (**node**), a direction at that node (**dir**)[1], a position in the queue (**place**), and has a particular destination address (**des**). We are interested in estimating the total waiting time for all traffic lights for each car until it arrives at the destination address given its current node, direction, place, and the decision of the light (red or green). We write $Q([node, dir, place, destination], action)$ to denote this value, which is sometimes shortly denoted as $Q([n, d, p, des], L)$. We write $V([node, dir, place, destination])$ to denote the average waiting time (without knowing the traffic light decision) for a car at $(node, dir, place)$ until it has reached its destination address.

4.2.4 Making a traffic light decision.

Given the current traffic situation, we can make a choice for each traffic node *node* as follows. All cars standing at those directions of the traffic node which are made green by the decision of the traffic node are examined and their individual advantages are summed up. The decision A_j^{opt} is finally selected by:

$$A_j^{opt} = \arg\max_{A_j} \sum_{i \in A_j} \sum_{(n,d,p,des) \in queue_i} Q([n, d, p, des], red) - Q([n, d, p, des], green)$$

Note how we calculate the overall advantage for deciding on the action: it sums over all cars at each traffic light set to green by the action of the traffic node (all the other traffic lights are red for all decisions). Since we sum overall waiting times (where waiting times are caused by the current traffic node, but also by next traffic situations), we have a global perspective. If the car is not on the first place and not waiting in the queue (i.e., the car can still drive onwards until it meets the queue), we do not let the car vote for the total waiting time, since the current decision of the traffic light does not affect the current transition of the car (the car always moves the same number of places further).

4.2.5 State transition probabilities and the reward function.

For computing the functions Q and V we use state transition probabilities. The probabilistic state transition function is given by a lookup table consisting of the probabilities: $P([node, dir, place, des], L, [new\_node, new\_dir, new\_place])$ where L denotes whether the light for $(node, dir)$ is red or green. This lookup table may seem to contain a huge number of elements, but that is not true: for all cars not standing at the first place of the queue, there can at maximum be few successors: staying in the same state or going to a next place for the same traffic light $(node, dir)$. Note also that the *des* variable for a car always stays the same. For cars standing at the first place, there can be at maximum 3 new road-lanes and some new-places which also means a small amount of possible state transitions. If a car crosses a traffic node, we compute its state transition probability to the new node, dir and place as that place where it is (at a certain moment) for the first time standing after traversing

[1] A traffic light is represented by combining the values for *node* and *direction*.

the junction (this is slightly different from our implementation in [Wiering, 2000] where the transition went to the next place a car would be standing in a queue).

We also compute probabilities $P(L|[node, dir, place, destination])$ which give the probability that the light is red or green for a car waiting at $(node, dir, place)$ with a particular destination. These probabilities are needed to compute the average waiting time $V([node, dir, place, destination])$. Finally, we use a reward function as follows:
$R([node, dir, place], L, [node, dir, place]) = 1$, if a car stays at the same place — it waits a time-step if it stays on the same place (note that this can be caused by the traffic light which is red or by too many cars on the next road-lane so that the first car cannot continue even when the light is green). Otherwise R=0 (the car can advance).

4.2.6 TC-1: Computing the V- and Q-functions.

For our system, TC-1 (traffic controller 1), we compute the Q-function as follows:

$$Q([n,d,p,des],L) = \sum_{(n',d',p')} P([n,d,p,des],L,[n',d',p']) \\ (R([n,d,p],L,[n',d',p']) + \gamma V([n',d',p',des])) \quad (2)$$

Where γ is the discount factor ($0 < \gamma < 1$) which ensures that Q-values are bounded. We compute the V-function using the Q-function and the probabilities of the light being green or red as follows:

$$V([n,d,p,des]) = \sum_{L} P(L|[n,d,p,des])Q([n,d,p,des],L) \quad (3)$$

Thus, if the probability of waiting is large (for which the immediate reward or cost is 1), the V-value and Q-values will be large. The V-value simply averages the Q-function using the different action probabilities. The Q- and V-functions are initialized to 0.0 values.

After each simulation step we compute the new transition probabilities and then we compute the V-values and Q-values by iterating using Real Time Dynamic Programming (RTDP) [Barto et al., 1995] which fixes the number of iterations. We could iterate until all values are exactly computed, but this would take a very long time. Therefore it may be better to iterate a small number of iterations (e.g., only 1). In our current experimental setup, we only update Q- and V-values of states where cars arrive during the current time-step. Although this does not recompute the complete value function, it is a very fast way of partially recomputing the desired functions.

TC-1 only uses locally available information and computes waiting times of cars without considering traffic situations at other traffic lights. It needs to know car-places for cars (which in realistic applications could be inferred using camera's mounted on the traffic lights, by measuring and communicating the distance to the traffic light, or by reliable inductive loops in the roads). Furthermore, the algorithm uses the destination address of cars which should be communicated to the traffic light.

4.2.7 Some notes on the behavior of the controllers.

If we analyze the learning equations and the voting mechanism, we can observe how the behavior of the adaptive traffic light controller will be:

- If many cars are waiting at a traffic light, they count more in the overall decision process than if only a single car has to wait. Therefore long queues are usually passed first.
- If the first car has a large advantage of exiting immediately, instead of entering a new long queue, this car may get a stronger vote since we expect its gain to be larger.
- If cars at some light have to wait a long time, the probability of waiting gets large. Note that if this probability goes to 1, the overall computed expected waiting time for the red light goes to infinity[2]. Since large waiting probabilities mean large expected waiting times and thus large votes against having to wait, the individual waiting times of cars are likely to be spread across the traffic lights of a traffic node. In this perspective, the traffic light controllers are very fair.
- The traffic light controllers are adapting themselves all the time. Therefore they can handle changing traffic flow patterns, although it may cost some time for the algorithm to adjust the parameters.

4.2.8 Adapting the system parameters.

For adapting the system, we only need to compute the state transition probabilities, since the reward function is fixed (standing still costs 1, otherwise the reward/cost is 0). First of all we compute $P(L|[node, dir, place, des])$, the probability that a light is red or green for a car with some destination at a specific place. For computing this probability we update counters after each simulation step: $C([node, dir, place, des], L)$ counts the number of times the light was red or green when a car was standing in the queue at the position determined by *place*. $C([node, dir, place, des])$ denotes the number of cars with some destination which have been standing in the queue at *place*. From this, we can compute the probability:

$$P(L|[node, dir, place, des]) = \frac{C([node, dir, place, des], L)}{C([node, dir, place, des])}$$

Furthermore, we have to compute the state transition probabilities for the first (or other cars). In the general version this is: $P([node, dir, place, des], L, [n, d, p])$. For cars which are not able to cross an intersection, this is quite trivial. There are only few possible state-transitions; a next place or the current place. State transitions are stored in variables: $C([node, dir, place, des], L, [n, d, p])$. If the car has made a transition to the next traffic light, it arrives at a new place at the next road-lane, and we also update the counter variables. To compute the requested probability, we normalize these counters by dividing each counter value by the total number of times a car was standing at a particular road-cell (with the specific destination of the car):

$$P([node, dir, place, des], L, [n, d, p]) = \frac{C([node, dir, place, des], L, [n, d, p])}{C([node, dir, place, des], L)}$$

4.2.9 Co-learning.

A nice feature of our car-based value functions is that they can be immediately used to select a path of traffic lights for a car to its destination address. Note that in a city there can be

[2]In case no discounting is used.

multiple (approximate) shortest paths from one starting point to a destination address. The normal (non co-learning) systems generate at each traffic light what the current options are to go from one traffic light to the next one in order to go to the destination address along one of the shortest ways. Then they select one of these randomly. Co-learning can be used to select among these shortest paths that path with minimal expected waiting time.

Suppose a car on the first place, planning to traverse the intersection, has destination des and is at $(node, dir)$. Then it generates a list of options at each traffic light, where each option is given by a pair (n, d), a new node and a new dir (thus a road-lane leading to possibly a next traffic light). Then we compute $Q(n, d)$-values to select the best next (n, d). For TC-1, we use $Q(n, d) = V(n, d, 1, des)$ to determine the next road-lane (n, d). Thus, we only compare the remaining average waiting time when the car would come to the first place at the next queue. Note that we do not use information about the number of cars in the next queue to estimate the place a car will enter a next queue. We want to examine possible advantages of including the next road's crowdedness in future research.

4.3 Discussion

As mentioned before, most traffic light controllers are fixed-cycle controllers, in which all alternative traffic lights settings get a particular time-interval for being green. In our approach, we use the actual situation to set the traffic lights, and therefore we have much more possibilities for optimizing the traffic light settings. Furthermore, our algorithm learns automatically, and does not require any manual tuning. This also leads to an adaptive system which can cope with changing traffic conditions. Since we use car-based value functions instead of traffic-light based value functions as used by Thorpe (1997), we do not need to use an approximation technique such as neural networks for learning the value function. Instead by using voting, we are able to get very accurate predictions of the consequences of different traffic light settings. Furthermore, our system is very fair; it never lets one vehicle wait all the time, not even if competing road-lanes are much more crowded. The fairness is a result of the learning and voting mechanism, and we did not need to design this ad-hoc. Finally, we can use the value functions for optimizing driving policies, where cars drive to minimize their waiting times, and traffic light controllers set the lights to minimize the waiting times as well. This co-learning system exploits the learned value functions and leads to very interesting co-operative multi-agent system dynamics.

We have not yet included green waves in our system, although we have included a simple technique for dealing with congested traffic. The bucket algorithm, which we use, propagates gains of one traffic light to the next lane, if the first car on a lane cannot drive onwards, because the next lane is full. This will lead to higher gain values for setting green lights to road-lanes that are full, and for which other cars have to wait if their light is set to green. This bucket technique is used in our current experiments, but needs more research to work fruitfully with our adaptive model-based RL algorithm.

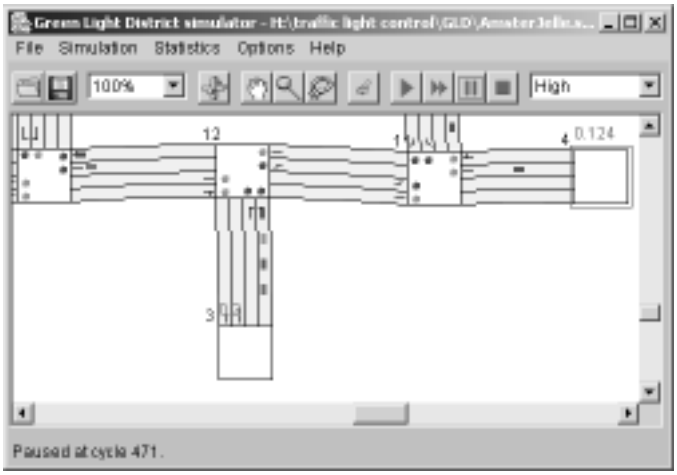

Figure 1: A screenshot from the Green Light District simulator. The image shows a junction with traffic lights, two edge-nodes, and some roads with cars. The number above the circled edge-node on the right indicates its spawning frequency.

5 Green Light District

We used the Green Light District (GLD)[3] traffic simulator for our experiments. GLD consists of an editor to define infrastructures (based on cellular automata), a Multi-Agent System (MAS) to run the simulation, and a set of controllers for the agents. The simulator has several statistical functions to measure performance. In the next section we will describe the simulator.

5.1 The Simulator

5.1.1 Infrastructures.

An infrastructure consists of roads and nodes. A road connects two nodes, and can have several lanes in each direction (see Figure 1). The length of each road is expressed in units. A node is either a junction where traffic lights are operational (although when it connects only two roads, no traffic lights are used), or an edge-node.

5.1.2 Agents.

There are two types of agents that occupy an infrastructure; vehicles and traffic lights. All agents act autonomously, following some simple rules, and get updated every time-step.

Vehicles enter the network at the edge-nodes. Each edge-node has a certain probability of generating a vehicle at each time step. Each vehicle that is generated is assigned a destination, which is one of the other edge-nodes. The distribution of destinations for each edge-node can be adjusted.

There are several types of vehicles, defined by their speed, length, and number of passengers. For our experiments we only used cars, which move at a speed of two units (or one or

[3]GLD is free, open-source software, and can be downloaded from {*http* : *//sourceforge.net/projects/stoplicht/*}

zero if they have to brake) per time step, have a length of two units, and have two passengers. The state of each vehicle is updated every time step. It either moves with the distance given by its speed, or stops when there is another vehicle or a red traffic light ahead. At a junction, a car decides to which lane it should go next according to its driving policy. Once a car has entered a lane, it cannot switch lanes.

Junctions can be occupied by traffic lights. For each junction, there are a number of possible ways of switching the lights that are safe. At each time-step, the traffic light controller decides which of these is the best. It can use information on waiting vehicles and their destination, and about other traffic lights to make this decision.

5.1.3 Statistics.

Data can be collected through several statistical functions. For each node, the number of road users that crossed, and the average time that a car has to wait can be recorded. Globally, the total amount of arrived road users, the average junction waiting time, the average trip waiting time (ATWT) and the total waiting queue length can be recorded. The waiting queue is the amount of vehicles that is generated, but cannot enter the infrastructure because the lanes from one or more edge-nodes are occupied.

5.2 The Controllers

Every junction is controlled by a traffic light controller (TLC) that decides on the best configuration of red and green lights. A TLC will only consider safe configurations, that is, configurations in which moving cars do not intersect. A TLC can share information with other controllers to improve global performance. GLD has several built in TLCs, and allows for custom TLCs. In the next subsections we describe the controllers that were used in our experiments.

5.2.1 TC-1.

The TC-1 algorithm uses the reinforcement algorithm described is section 4.2. TC-1 computes the gain of setting (two) lights to green, and chooses the configuration with the highest gain. To compute the gain, TC-1 compares expected trip waiting times for when the current light is set to green and red for the road user. To this end it needs information on the destination of the car. The discount factor used by the algorithm was set to 0.9.

5.2.2 TC-1 Destinationless.

TC-1 Destinationless is a simplified version of TC-1 that does not use destinations in its computations.

5.2.3 TC-1 Bucket.

TC-1 Bucket is TC-1 extended with the Bucket algorithm that is described below.

5.2.4 TC-1 Co-learning.

Co-learning can be used with TC-1 and with TC-1 Bucket, but not with TC-1 Destinationless, since it requires information about vehicle destinations. We only use co-learning with TC-1.

5.2.5 The Bucket algorithm.

We developed the bucket mechanism for optimizing traffic flow in cities of very high traffic densities. It can easily be applied to any traffic light controller algorithm that calculates gain values for individual traffic lights per node. The basic thought underlying the mechanism is that each traffic light from which no movement is possible due to overfull destination lanes communicates part of its summed gain values there to stimulate movement. Each traffic light sums the gain values calculated by the traffic controller algorithm into its bucket. This value is used to determine the optimal configuration of green traffic lights, instead of the originals calculated by the TLC. In case the light is green but the road users cannot move due to full destination lanes, we siphon over part of the bucket value into the bucket of the destination traffic light. Once a road user passes a traffic light the bucket value is decreased by $1/n$, where n is the number of road users on the lane. Though this seemingly simple mechanism is based on purely local communication it creates directed global flow of traffic. The usage of the bucket principle also provides a measure against indefinite waiting of small numbers of road users for crowded lanes at the same node; due to the summation in the bucket mechanism their bucket value will ensure that they will be able to move on.

5.2.6 Best First.

The best first controller chooses that option where the most road users waiting for a junction can move. Thus it just counts each car that can drive on, and chooses the traffic light setting such that most cars can drive further.

5.2.7 ACGJ-3.

ACGJ-3 was the first traffic light controller algorithm we paired with the bucket mechanism. The gain value for a traffic light is calculated by summing over all waiting road users, multiplying the road user weight (i.e., number of passengers, or fixed value) with a length-factor. By choosing this factor $f < 1$ we force the algorithm to value most the first road users, whilst by choosing $f > 1$ we weigh longer waiting queues exponentially. In the experiments presented in this paper we used the ACGJ-3 algorithm (with $f = 1$), resembling the Best First algorithm paired with the bucket mechanism.

5.2.8 Relative Longest Q.

The relative longest queue is the queue with the maximum ratio of waiting road users versus length of the road. Because completely filled up roads get priority over roads where many people are waiting, but are not full, this algorithm may avoid jammed traffic.

5.2.9 Random.

Random is the simplest controller we used. As one might guess, it decides totally random. Results of the random controller do not show in our experimental results, since we only used it as a worst-case scenario (it would always lead to completely congested traffic in our experiments).

5.3 Driving Policies

At a junction, a car has to decide which lane to go to next in order to reach its destination. The way this decision is made can be changed by using different driving policies.

5.3.1 Shortest Path.

For all junctions, there is a list of shortest paths to every destination. All paths that are no more than ten percent longer than the shortest path are in this list. When a decision has to be made, one of these shortest paths is selected randomly.

5.3.2 Co-learning.

The "theory" of co-learning was already explained in section 4.2. In GLD, Co-learning is implemented as follows. When a driving decision has to be made, all the shortest paths to the destination are collected. The shortest path with the lowest co-learn value is selected for the car intending to cross the intersection. The selected path should be the path to the destination with minimal expected waiting time.

6 Results

We compared the TC-1 reinforcement learning algorithms to other traffic light controllers using the GLD traffic simulator. All tests were performed with reasonably high traffic load. We first experimented with different amounts of cars entering the network each cycle by adjusting the spawning frequencies. We then set them as close as possible to the point where traffic would get jammed, the congestion point.

For each test, we ran ten simulations of 50.000 cycles each. The results show the average over those ten simulations. The measure we used to compare performance is the average time a car has to wait during a trip, the ATWT.

6.1 Experiment 1: A Large Infrastructure

6.1.1 Experiment.

For our first series of experiments, we used the grid-like infrastructure depicted in Figure 2. With 12 edge nodes and 16 junctions, 15 of which have traffic signs, and 36 roads, 4 lanes wide each, it is quite a big structure. When totally filled up, it could contain about 1500 vehicles. All edge nodes were set at the same spawning frequencies, and for each edge node, all other nodes were given an equal chance of being the destination of a new car.

We tested all controllers under the same, high traffic load. All edge-nodes were set at the same spawning frequencies, and destinations were distributed equally. Preliminary experiments showed that a maximum load is reached when all spawning frequencies are set to 0.4. At this load, traffic gets jammed when the simpler (for example random) controllers are used, but the more complex controllers keep traffic flowing. At higher levels, no controller can avoid congestion, therefore we set all spawning frequencies to 0.4 in this experiment.

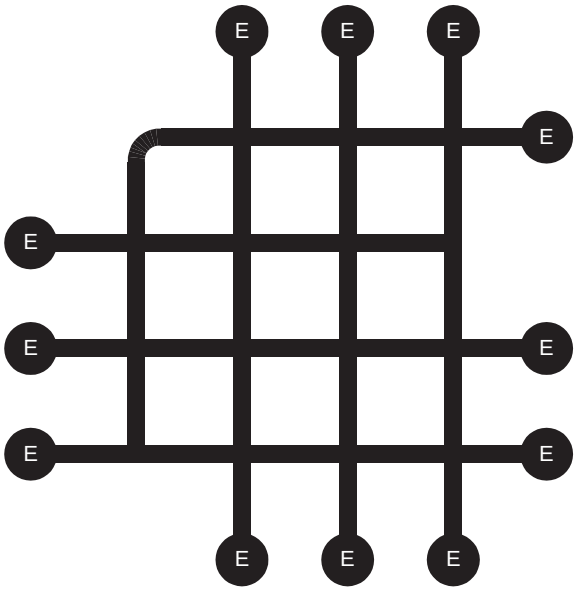

Figure 2: Infrastructure for the first experiment. Nodes marked with E represent edge-nodes. All intersections are controlled by traffic lights, and all edges represent roads with two lanes in each direction. At each edge node, 0.4 cars are generated each cycle.

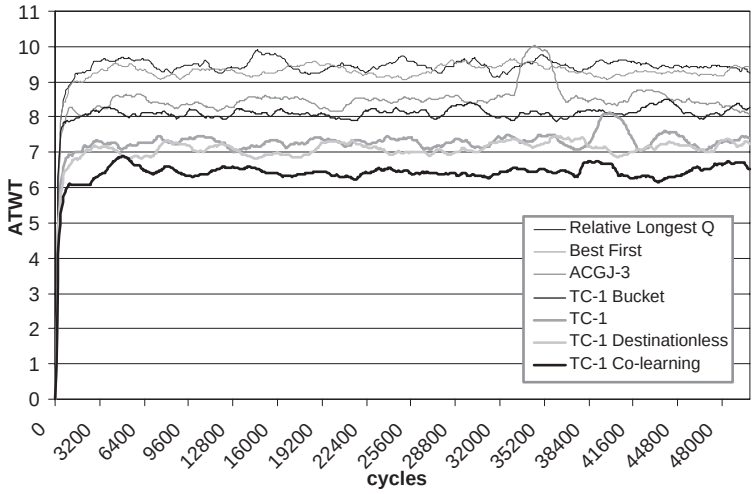

Figure 3: Results of the first experiment. Each result is an average of ten simulations.

Controller		ATWT
1	TC-1 Co-learning	6.46
2	TC-1 Destinationless	7.14
3	TC-1	7.40
4	TC-1 Bucket	8.20
5	ACGJ-3	8.46
6	Best First	9.27
7	Rel Longest Q	9.46

Table 1: Comparison of the controllers on the infrastructure of Figure 2. the ATWT shown is an average over the last 10.000 cycles of all ten experiments.

6.1.2 Results.

As shown in Figure 3, the TC-1 algorithms outperform all fixed algorithms. TC-1 with co-learning clearly performs best, with an average ATWT of 6.46 during the last 10.000 cycles (see Table 1). TC-1 Destinationless, the simplified version of TC-1, performs surprisingly well, and seems to outperform TC-1 when co-learning is not used. The TC-1 bucket algorithm is the worst performing of the TC-1 algorithms, but still outperforms ACGJ-3, which is a best first algorithm with a bucket. Note that the best non-adaptive controller (ACGJ-3) has an average waiting time which is more than 30% larger than the ATWT of the best TC-1 algorithm.

6.2 Experiment 2: Co-learning

In this experiment we use a simple infrastructure to study whether and how the use of co-learning as a driving policy can improve performance of a traffic network.

6.2.1 Experiment.

For this experiment, we designed a simple infrastructure, where drivers have a choice between a crowded and a less crowded route. The infrastructure is shown in Figure 4. In this network, we created two streams of traffic, one moving from E2 to E3, and one moving from E1 to E4. Traffic from E1 to E4 has a choice between two routes of equal length, route A (via nodes 1, 2, 3, and 6) or route B (via nodes 1, 2, 4, 5, and 6). Route B is the busier route, since part of it is occupied by traffic moving from E2 to E3, so vehicles traveling through route B can be expected to have higher waiting times than those that take route A. Preliminary experiments showed that traffic was crowded, but kept flowing, when the spawning frequencies were set to 0.6 for E1, and 0.54 for E2. We performed experiments using the TC-1 algorithm with a shortest path driving policy, and with the co-learning driving policy.

Driving Strategy	ATWT
Co-learning	0.19
Shortest Path	0.40

Table 2: Results of experiment two: Average ATWT during the last 10.000 cycles.

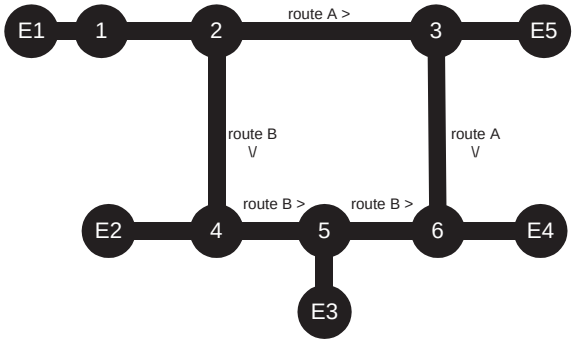

Figure 4: Infrastructure for the second experiment. Nodes marked with E represent edge-nodes, and nodes 2 till 6 are junctions with traffic lights. All edges represent roads with two lanes in each direction. Traffic from E1 to E4 could choose route A or route B, which have the same length, but route B is busier due to heavy traffic from E2 to E3.

6.2.2 Results.

The results are shown in Figure 5 and Table 2. They show that co-learning is an improvement on the driving policy, and results in 50% lower waiting times when compared with a random shortest path policy. What does not show in these results, but did show in observing the experiments is the actual driving behavior. In the beginning of each experiment, during which the controllers for the traffic lights have not learnt much yet, vehicles from E1 to E4 are divided between route A and B, but within several hundred cycles, all follow route A. The fact that there still is some waiting time when co-learning is used is caused by the fact that, in choosing an optimal route, vehicles in node 1 in Figure 4 select the same optimal next lane. This causes some waiting time due to the high spawning frequency.

6.3 Experiment 3: An Inner City

Our aim for the third experiment was to measure performance in a city-like structure. We designed an infrastructure that represents the ring-road around a city, the access to the city, and the access to highways.

6.3.1 Experiment.

The infrastructure that we used for our third experiment is shown in Figure 6. The inner edge-nodes represent access to the city center, and the outer edge-nodes are the connections to the approach roads. The network is quite big, with a total of 615 units length of two-lane, two-way roads. We again determined a traffic load near saturation by experimenting with different spawning frequencies. For this infrastructure, this point occurred when all spawning frequencies are set to 0.4.

6.3.2 Results.

The results are shown in Table 3 and Figure 7. The TC-1 algorithms again outperformed the others, but TC-1 with co-learning is no longer the best. TC-1 with the normal shortest

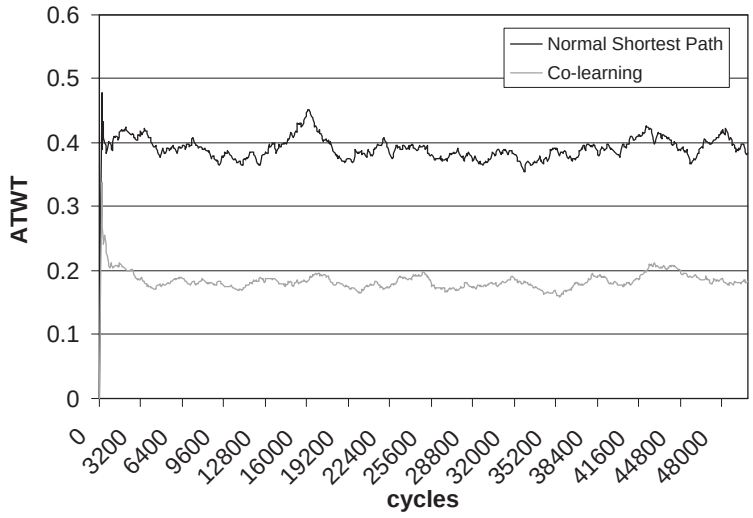

Figure 5: Average trip waiting time of TC-1 when drivers take the shortest path, and when drivers use Co-learning. Results are an average of ten simulations on the infrastructure depicted in Figure 4. Co-learning improves performance because vehicles follow route A rather than the busier route B.

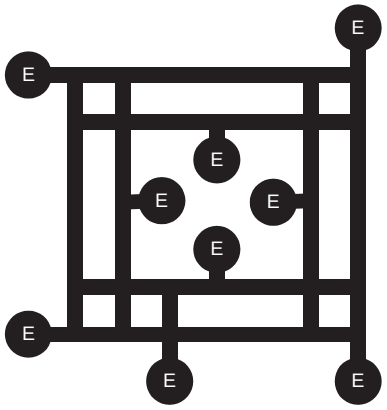

Figure 6: Infrastructure for the third experiment. The inner edge-nodes represent a city center, surrounded by ring roads. Traffic from outside the city can either drive around or enter the city center. Traffic coming from the center always leaves the city. All intersections feature traffic lights.

	controller	ATWT
1	TC-1 Destinationless	2.67
2	TC-1	2.68
3	TC-1 Co-learning	2.82
4	TC-1 Bucket	3.25
5	ACGJ-3	3.57
6	Best First	4.23
7	Rel Longest Q	4.75

Table 3: Results of the third experiment.

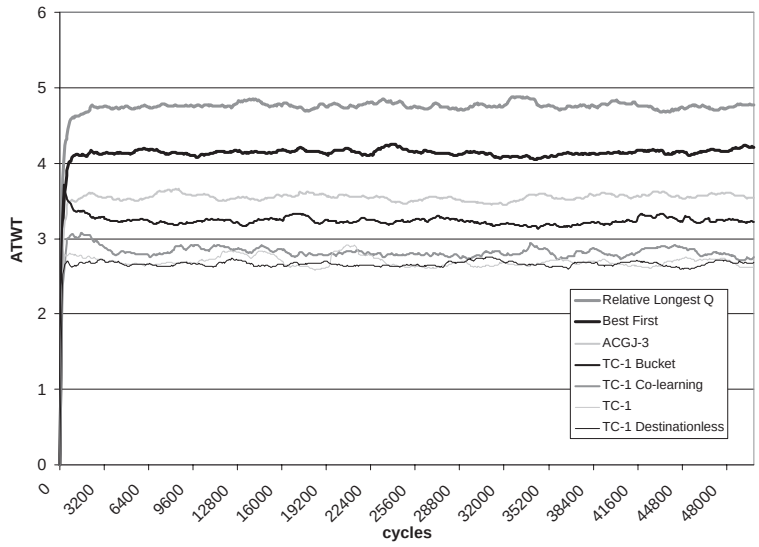

Figure 7: Results of the third experiment.

path driving policy performs better, indicating that the effect of co-learning depends on the infrastructure. In this particular structure, where there are few origins and destinations, but always multiple routes, choosing between routes randomly may result in an equal distribution of vehicles. When co-learning is used, all drivers with the same origin and destination choose the same route, and that route might get more saturated. Another interesting result is the fact that the TC-1 destinationless algorithm, which is a simplified version of TC-1, performs as well as the TC-1 algorithm. Note that the best non adaptive controller (again ACGJ-3) has an ATWT which is more than 25% larger than the ATWT of the best RL algorithm.

6.4 Discussion

In our experiments, we have seen that our adaptive controllers clearly outperform fixed controllers. The simulator currently makes use of some simplifications, since we do not model acceleration and deceleration. It could quite easily be extended to offer such a higher level of detail, and adaptive algorithms could benefit from this by incorporating actual speed infor-

mation of road users in their calculations.

We have seen that co-learning may work well, but more research can be done to improve the idea of co-learning even more. Now we use the first place, but in our current set-up we could also have used the V-value of the last place on a roadlane. Furthermore, in previous work [Wiering, 2000] we communicated how many cars are standing on each next lane to choose the next lane. This may also improve the results of co-learning.

Finally, we have not compared our adaptive methods with other adaptive traffic light controllers. We want to integrate other traffic light controllers in our system and evaluate these.

7 Conclusions

In this article we first showed that traffic control is an important research area, and its benefits make investments worthwhile. We described how traffic can be modelled, and showed the practical use of some models. In section 3 we explained reinforcement learning, and showed its use as an optimization algorithm for various control problems. We then described the problem of traffic light control and several intelligent traffic light controllers, before showing how car-based reinforcement learning can be used for the traffic light control problem. In our approach we let cars estimate their gain of setting their lights to green and let all cars vote to generate the traffic light decision. Co-learning is a special feature of our car-based reinforcement learning algorithm that allows drivers to choose the shortest route with lowest expected waiting time.

We performed three series of experiments, using the Green Light District traffic simulator. We described how this simulator works, and which traffic light controllers were tested. The experiments were performed on three different infrastructures. The first experiment, which uses a large grid, shows that reinforcement learning is efficient in controlling traffic, and that the use of co-learning further improves performance. The second experiment shows that using co-learning vehicles avoid crowded intersections. This way, vehicles avoid having to wait, and actively decrease pressure on crowded intersections. The third experiment shows that RL algorithms on more complex and city-like infrastructure again outperform the fixed controllers by reducing waiting time with more than 25%. The third experiment also shows that in some situations a simplified version of the reinforcement learning algorithm performs as well as the complete version, and that co-learning not always increases performance.

7.1 Further Research

Although the reinforcement learning algorithm presented here outperforms a number of fixed algorithms, there are several improvements that could be researched. For example, we can use communication between road-lanes to make green waves possible, and let estimated waiting times depend on the amount of traffic on the next road-lane.

The co-learning driving policy might be improved as well. The current implementation suffers from saturation and oscillation. Because all drivers on a route choose the optimal lane, this lane might become crowded. Only when the performance of such a lane decreases because of this crowding, drivers will choose another lane. A less greedy form of co-learning might prevent this effect.

Although the bucket algorithm works well for the fixed algorithms, it did not work well together with the RL algorithms. We have to study this more carefully, since the bucket

algorithm, when designed well, may help in creating green waves in very crowded traffic conditions.

The simulator might be refined as well, to allow for comparison with other research. Refinements could include more complex dynamics for the vehicles and other road users, as well as an implementation of fixed-cycle traffic-light controllers.

References

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

[Baxter et al., 1997] Baxter, J., Tridgell, A., and Weaver, L. (1997). Knightcap: A chess program that learns by combining TD(λ) with minimax search. Technical report, Australian National University, Canberra.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.

[Ben-Akiva et al., 2003] Ben-Akiva, M., Cuneo, D., Hasan, M., Jha, M., and Yang, Q. (2003). Evaluation of freeway control using a microscopic simulation laboratory. *Transportation research Part C: emerging technologies*, 11-1:29–50.

[Brauer and Weiss, 1997] Brauer, W. and Weiss, G. (1997). Multi-machine scheduling — a multi-agent learning approach. Technical report, Technische Universität München.

[Broucke and Varaiya, 1996] Broucke, M. and Varaiya, P. (1996). A theory of traffic flow in automated highway systems. *Transportation research Part C: emerging technologies*, V4:181–210.

[Choi et al., 2002] Choi, W., Yoon, H., Kim, K., Chung, I., and Lee, S. (2002). A traffic light controlling FLC considering the traffic congestion. In Pal, N. and Sugeno, M., editors, *Advances in Soft Computing — AFSS 2002, International Conference on Fuzzy Systems*, pages 69–75.

[Crites and Barto, 1996] Crites, R. H. and Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1017–1023. The MIT Press.

[Dia, 2002] Dia, H. (2002). An agent-based approach to modelling driver route choice behaviour under the influence of real-time information. *Transportation Research Part C: Emerging Technologies*, 10-5/6:331–349.

[Emmerink et al., 1996] Emmerink, R. H. M., Nijkamp, P., Rietveld, P., and van Ommeren, J. N. (1996). Variable message signs and radio traffic information: an integrated empirical analysis of drivers' route choice behaviour. *Transportation Research Part A: Policy and Practice*, 30-2:135–153.

[EPA98, 1998] EPA98 (1998). Assessing the emissions and fuel consumption impacts of intelligent transportation systems (ITS). Technical Report United States Policy EPA 231-R-98-007, Environmental Protection Agency Washington.

[Findler and Stapp, 1992] Findler, N. and Stapp, J. (1992). A distributed approach to optimized control of street traffic signals. *Journal of Transportation Engineering*, 118-1:99–110.

[Helbing et al., 2002] Helbing, D., Hennecke, A., Shvetsov, V., and Treiber, M. (2002). Micro- and macro-simulation of freeway traffic. *Mathematical and Computer Modelling*, 35-5/6:517–547.

[Horowitz and Varaiya, 2000] Horowitz, R. and Varaiya, P. (2000). Control design of an automated highway system. In *Proc. IEEE, vol. 88*, pages 913–925.

[Jin and Zhang, 2003] Jin, W. L. and Zhang, H. M. (2003). The formation and structure of vehicle clusters in the payne-whitham traffic flow model. *Transportation Research Part B: Methodological*, 37-3:207–223.

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

[Ledoux, 1996] Ledoux, C. (1996). Application of neural networks to long term prediction of queue length at an urban traffic junction. Technical Report 1996/17, IBP-Laforia.

[Lee et al., 1995] Lee, J., Lee, K., Seong, K., Kim, C., and Lee-Kwang, H. (1995). Traffic control of intersection group based on fuzzy logic. In *Proceedings of the 6th International Fuzzy Systems Association World Congress*, pages 465–468.

[Levinson, 2003] Levinson, D. (2003). The value of advanced traveler information systems for route choice. *Transportation Research Part C: Emerging Technologies*, 11-1:75–87.

[Lighthill and Whitham, 1955] Lighthill, M. J. and Whitham, G. B. (1955). On kinematic waves: Ii. a theory of traffic flow on long crowded roads. *Proceeding of the Royal Society A*, 229:317–345.

[Littman and Boyan, 1993] Littman, M. and Boyan, J. (1993). A distributed reinforcement learning scheme for network routing. In Alspector, J., Goodman, R., and Brown, T., editors, *Proceedings of the First International Workshop on Applications of Neural Networks to Telecommunication*, pages 45–51, Hillsdale, New Jersey. http://www.cs.duke.edu/ mlittman/docs/routing-iwannt.ps.

[Liu et al., 2002] Liu, H. L., Oh, J.-S., and Recker, W. (2002). Adaptive signal control system with on-line performance measure. In *81st Annual Meeting of the Transportation Research Board*.

[Mataric, 1994] Mataric, M. J. (1994). *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts institute of Technology. http://www.cs.bham.ac.uk/~sra/People/Mno/Mataric/#Interaction.

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

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

[Nagel and Schreckenberg, 1992] Nagel, K. and Schreckenberg, M. (1992). A cellular automaton model for freeway traffic. *J. Phys. I France*, 2:2221–2229.

[Rechenberg, 1989] Rechenberg, I. (1989). Evolution strategy: Nature's way of optimization. In Bergmann, editor, *Methods and Applications, Possibilities and Limitations*, pages 106–126. Lecture notes in Engineering.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.

[Schaerf et al., 1995] Schaerf, A., Shoman, Y., and Tennenholtz, M. (1995). Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, 2:475–500. http://www.cs.washington.edu/research/jair/contents/v2.html.

[Stone and Veloso, 1997] Stone, P. and Veloso, M. (1997). Multiagent systems: A survey from a machine learning perspective. Technical report, Carnegie Mellon University.

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

[Sutton, 1996] Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, 8.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT press, Cambridge MA, A Bradford Book.

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

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

[Tan, 1993] Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337.
http://www.cs.brandeis.edu/~aeg/papers/tan.ML93.ps.

[Tavladakis and Voulgaris, 1999] Tavladakis, K. and Voulgaris, N. C. (1999). Development of an autonomous adaptive traffic control system. In *ESIT '99 - The European Symposium on Intelligent Techniques*.

[Ten-T expert group on ITS, 2002] Ten-T expert group on ITS for road traffic management (2002). deployment of intelligent transport systems on the trans-european road network. Technical report, European Commision.

[Tesauro, 1992] Tesauro, G. (1992). Practical issues in temporal difference learning. In Lippman, D. S., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 4*, pages 259–266. San Mateo, CA: Morgan Kaufmann.

[Thorpe, 1997] Thorpe, T. (1997). Vehicle traffic light control using sarsa. Master's thesis, Department of Computer Science, Colorado State University.

[Thorpe and Andersson, 1996] Thorpe, T. L. and Andersson, C. (1996). Traffic light control using sarsa with three state representations. Technical report, IBM corporation.

[Thrun, 1992] Thrun, S. (1992). Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie-Mellon University.

[Wahle and Schreckenberg, 2001] Wahle, J. and Schreckenberg, M. (2001). A multi-agent system for on-line simulations based on real-world data. In *Proc. of the Haiwai International Conference on System Science (HICSS)*. IEEE Computer Society.

[Watkins, 1989] Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England.

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

[White Paper, 2001] White Paper (2001). European transport policy for 2010: time to decide.

[Wiering et al., 1999a] Wiering, M., Krose, B., and Groen, F. (1999a). Learning in multi-agent systems. Technical report, University of Amsterdam.

[Wiering, 1999] Wiering, M. A. (1999). *Explorations in Efficient Reinforcement Learning*. PhD thesis, University of Amsterdam.
http://carol.wins.uva.nl/~wiering/publications.html.

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

[Wiering et al., 1999b] Wiering, M. A., Salustowicz, R. P., and Schmidhuber, J. (1999b). Reinforcement learning soccer teams with incomplete world models. *Artificial Neural Networks for Robot Learning. Special issue of Autonomous Robots*, 7(1):77–88.