Controlling traffic signals

Justus Piater

December 22, 1995

Abstract

An RL-based traffic light control system was developed that uses no more information than I believe to be available in a real environment. Sparse information about the environment is used to update an internal functional model, on which Q-learning is performed. Real actions are selected with respect to the model. The system learns to effectively control traffic at a road intersection. Major limitations are due to the model-based approach used.

1 Introduction

Traffic light control systems currently use fixed policies. The most sophisticated ones use inductive loops to detect the presence of a waiting car. An incoming low-traffic lane is only set to green if a car is waiting.

The approach presented here uses these inductive loops to infer parameters about the traffic structure, updates an internal parametric functional model of the intersection, and performs Q-learning on this model to develop an improved policy for the real environment.

An inductive loop is assumed to be capable of

- telling whether or not a car is waiting at a given instance of time, and
- counting cars as they pass over it.

For simplicity, one isolated intersection is modeled. Arrival times of cars are generated randomly. A separate lane with its own inductive loop is assumed for each direction, as illustrated in Fig. 5. A car may only pass if its lane has explicitly been set to green. Possible combinations of light settings are hard-coded into the system.

2 Situations, Actions, Rewards

An *action* of the agent is to set the lights to a particular combination of settings. The new setting may be identical to the old one.

After an action a_t has been taken, the intersection is observed for a fixed period of time.[1] This observation yields a situation vector s_{t+1} and a reward r_{t+1}.

A *situation* is modeled as a vector of $l+1$ elements, where l is the number of lanes. The sum of the waiting times of the individual cars for each lane are given, as well as the current light setting.

The *reward* is given by the negative sum of the squared waiting times of all cars in all lanes. The target is the infinite-horizon discounted return.

Action values are represented as one table for each action. Summed waiting times are discretized into non-overlapping cells. Care was taken to trade off between accuracy and dimensionality. A hash-table representation of the tables is used.

3 The simulator

The simulator used to simulate a lane of the intersection (the *environment*) works as follows:

When the lane is observed, random arrival times of cars are generated. They are drawn from a normal distribution with given standard deviation and mean

[1] Hence, this system operates in discrete time steps. It would also make sense to model an intersection as a discrete event system.

at the next expected arrival time. If the $(i+1)$st arrival time is less than the ith, then it is set equal to the ith. This simulates a slow car followed by a potentially faster one and results in rather irregular arrivals, if the standard deviation is large. Arrival times thus generated are appended to a linked list of waiting cars.

If the observed lane is set to green, simulated time is set to the beginning of the observation interval. If it was red during the previous interval, time is advanced by a constant called "yellow duration". Then, cars are taken from the head of this list. For each car, simulated time is advanced by the amount of time it takes a car to pass, or to the time of the arrival of the passing car, whichever is later.

This procedure is repeated until the observation interval is over, or until the queue is empty. The simulated clock is then set to the time at the end of the observation interval.

4 The model-based approach

4.1 Motivation

In a real environment, no information about the actual waiting times is available. Therefore, a model is used to fill in this information which is required for both the situation description and the reward computation.

Since the situation vector contains no information about traffic densities, the action value function is changed as the traffic parameters change. This leads to a second argument:

To prevent a change of traffic densities from causing inappropriate behavior of the controller, learning is done inside the model. It is assumed that this model can be updated effectively.

Given a sufficiently accurate model, no exploration needs to be done in the environment. If real observations are used to update the model, a lot of exploration can be done using the model until the next real action is due. At that time, hopefully, the action value function is accurate enough to come up with a reasonable action.

4.2 Operation

The interaction of the RL system with model and environment is shown in Fig. 1.

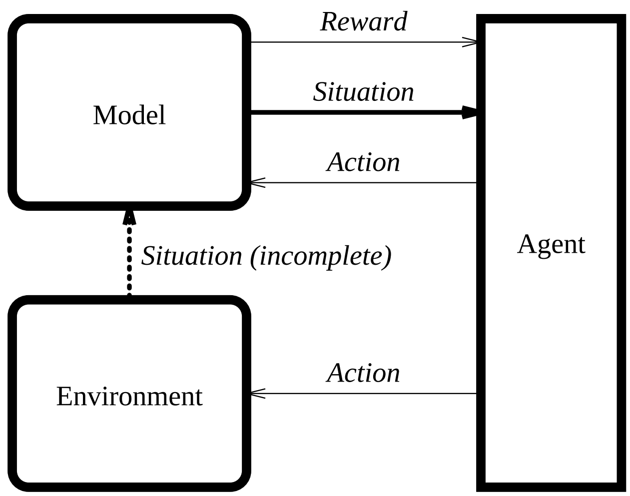

Figure 1: Structure of the model-based RL system and its interaction with the environment.

After observing the real environment for one period of time, this information is used to update the car arrival frequency parameter of the model (Fig. 2). The state of the model is reverted to what is believed to be the current state of the environment (Fig. 3). This state is given for each lane by

- the car arrival frequency f
- the arrival time τ_c of the first car in the line.

Then, ϵ-soft Q-learning (without eligibility traces) is performed on the model for a fixed number of actions. Whenever the sum of squared wait times exceeds a given threshold, the model is again reverted to the current state of the environment, and Q-learning continues. After completing the given number of model actions, the next real action is chosen (greedily) according to the current model, and the environment is observed again.

The repeated model trials starting at the current real state can be viewed as a form of planning. The idea behind this is that the agent tries to improve its policy around the current situation, where an improvement is most likely to influence the next real action.

No learning takes place across trials, ie. whenever a real action is taken or the model wait time threshold is exceeded, a new trial begins. The same threshold applies to the real environment as well: A real trial is interrupted when the number of waiting cars grows without bounds. Due to the tabular representation, the agent cannot discriminate between waiting times above a certain threshold.

In a real application, observation of the environment and model trials are to be run in parallel. The number of model actions per real action should not be fixed, but bounded by the duration of a real observation.

The full details of the model are too complex to be presented here. An outline of the basic steps is shown in Fig. 4. Some additional heuristic steps are performed to ensure consistency. This procedure is far from perfect, but appeared to work reasonably well.

For each lane do:

- If the queue was emptied during the last observation period (lasting from τ_0 to τ_1), set τ_e to this time.
- If it was green, then update the estimate of the arrival frequency f as follows:
  - target frequency:
    $$f_t := \frac{\text{\# of cars passed}}{\tau_1 - \tau_0}$$
  - If the queue was not emptied, then
    $$f_t := f_t \cdot \beta_f^{\frac{(\tau_1 - \tau_e)}{(\tau_1 - \tau_0)}}$$
  - new frequency:
    $$f := \min\left\{ f + \alpha_f (f_t - f), \frac{1}{t_{\text{pass}}} \right\}$$
    where t_{pass} is the time needed for a car to pass.

Figure 2: Updating the model from the environment.

5 Experiments

5.1 Sample intersections

Three different intersections were modeled which will be called *small*, *medium* and *large* respectively. They are illustrated in Fig. 5.

small: Two one way roads crossing each other. No turning is allowed. This is a minimal environment with just two lanes.

medium: Three meeting roads. All six lanes and all four light settings are modeled.

large: Two roads crossing each other. All twelve lanes and eight light settings are modeled (all but the setting where all four right turns are allowed).

An interactive framework allows to change various parameters at any time; in particular, the car arrival rates can be changed individually for each model or environment lane. In a typical experiment, the arrival rates of the environment lanes are set to some unequal values. As the system runs, the agent updates its model and simultaneously learns a policy by performing Q-learning on the model.

5.2 Performance graphs

Examples of performance graphs for the small and medium intersections are shown in Figs. 6 and 7. The term *cycle* here denotes one state-action-backup sequence.[2] It can easily be seen that the agent learns to control the traffic effectively. When the traffic parameters are radically changed (at real cycle 1000), it takes a while to adjust the policy. This extends over a number of real cycles because the model has

[2] In fact, only one backup is performed after each observation since plain Q-learning ($\lambda = 0$) is used.

For each lane do:

- If real cars are waiting, but the model sum wait time is less than half the observation period, assume the arrival of one car in the middle of this period.
- If no real cars are waiting, clear the model lane as well.

Figure 3: Reverting the model to the state estimate of the environment.

to be adapted. During this period the system has to be reset a few times because the agent does not cope with the traffic.[3] Note the increase in the model waiting time at the start and after the parameter change, where the the real waiting times decrease.

These graphs were radically smoothed[4] because huge outliers would otherwise hide any trends.

For comparison, Fig. 8 shows a longer run with fixed parameters without smoothing. As experiments showed, performance did not improve significantly beyond 2000 real training cycles.

It is interesting to note (but not surprising) that only a small fraction of the state space is actually visited. During the first few cycles, many new cells are generated. Effective learning starts when this process slows down. After the change of the arrival rates, once again a number of new cells is generated.

In the small intersection (Fig. 6), the waiting times increase at the end of the interval shown. As experiments have confirmed, this is pure chance: Apparently the system reached a state which it had not encountered before, and new Q cells were generated. After a few hundred cycles, the waiting times went down again and remained stable for another 2000 cycles. Note also the generation of two (apparently important) new cells at the "bump" around cycle 1600.

The huge rise in the model wait times (between 100 and 300 real cycles) is an artifact caused by inconsistencies of the model.

For each lane do:

- observation interval $d = \tau_1 - \tau_0$
- # cars waiting $n_w = f \cdot d$
- If it is green, then do
  - If it was not green before, then
    $$d := d - \text{(yellow duration)}$$
  - # cars passed $n_p = \frac{d}{t_{\text{pass}}}$
  - $n_w := \max\{0, n_w - n_p\}$
- Computation of the summed waiting time:
  $$f\int_{\tau_c}^{\tau_1}(\tau_1 - \tau)d\tau = \frac{f}{2}(\tau_1 - \tau_c)^2$$
- Computation of the summed squared waiting time:
  $$f\int_{\tau_c}^{\tau_1}(\tau_1 - \tau)^2 d\tau = \frac{f}{3}(\tau_1 - \tau_c)^3$$

Figure 4: Outline of a model observation.

Model wait times tend to be lower than real ones by about factor two. Nevertheless, they are accurate enough to control the system since relative differences are more important than absolute magnitudes.

5.3 Large intersection

The large intersection was harder to deal with for several reasons, e.g.

- The huge state space slows down learning. In fact, it was not possible to perform averaging of learning curves as this would have required several days.
- It is not obvious what nontrivial arrivial frequency combinations are feasible.

[3] It would be interesting to test whether it could handle *gradual* changes.

[4] Sliding average with boundary values replicated outside the range of the independent axis.

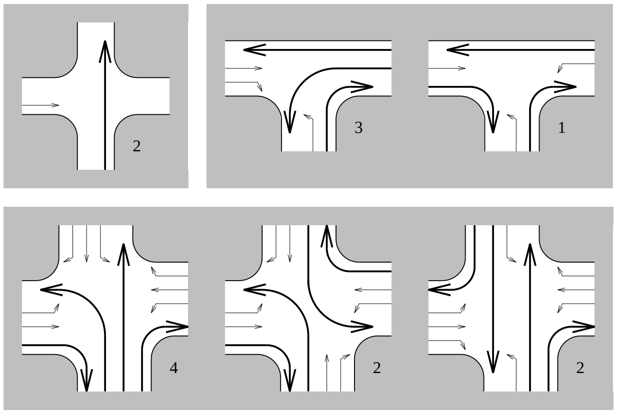

Figure 5: The three intersections modeled (top: small and medium; bottom: large). Bold arrows indicate lanes that are set to green. Prototypes of all possible light settings are shown. The digits give the number of different settings of a prototype.

Figures 9 and 10 show sample performance curves of a run where the model parameters were set to match the simulator ones, and model adaptation was turned off. Despite these advantages, it took clearly longer and a lot more Q value cells to converge.

To rule out that the arrival rates were trivial, a random policy was run. The squared waiting times virtually exploded.

For comparison, Fig. 11 shows a sample run where the model arrival rates had to adapt from initial default values. It took the system a lot longer to improve, and even after 10000 real cycles lots of new Q cells are generated. While in the end, the average squared waiting times are roughly the same as in the run with fixed model parameters, many outliers disturb the scene. It can be expected that the system will further improve with more cycles.

5.4 Sample policy

It is hard to illustrate the behavior of the policy.[5] For the small intersection, a tabular representation is still readable (Tab. 1). It can be seen that the learned policy adequately controls this simple intersection.

5.5 Learning Behavior

When the system starts with an empty intersection, a few action values are learned for the case of few cars. At first, the policy is inappropriate; therefore, huge numbers of cars accumulate at the busier lanes.[6] When the aforementioned threshold is reached, a new trial is started.

As the system learns to keep the sum squared wait time within bounds, this threshold is never reached again, and the trial extends to infinity. Hence, the action values are improved to a great accuracy—as accurate as the fixed step size parameter allows—in a small subsection of the state space. Typically, this subsection does not include the empty crossing because it has only been visited a small number of times; neither does it include unusual situations involving peak waiting times. If such a situation is encountered, the waiting times are likely to explode even further. This probably accounts for many of the outliers observed during the test runs.

These phenomena make it hard to design appropriate test procedures for the infinite horizon case. To improve this, a proper concept of trials should be defined and used for training.

Another cause of outliers are model inconsistencies which are introduced when the model is reverted to the environment.

[5] All protocol functionality is there—all I need is a volunteer to implement an animated visualization...

[6] If all lanes carry low traffic, no learning is evident since a random policy does just fine.

	Lane 0					*Lane 1*				
	Env.		*Model*			*Env.*		*Model*		
cyc	*wt*	*n*	*wt*	*n*		*wt*	*n*	*wt*	*n*	
600	0	0	0	4	•	16	1	6	1	∘
601	0	0	0	0	∘	21	1	13	1	•
602	0	0	0	3	•	0	0	0	0	∘
603	0	0	0	4	•	6	2	2	1	∘
604	16	7	3	0	•	19	3	6	1	∘
605	24	8	2	0	∘	34	3	13	2	•
606	64	8	21	3	•	0	0	0	0	∘
607	68	8	23	3	•	0	0	0	1	∘
608	26	5	9	2	•	1	1	2	1	∘
609	23	5	2	0	•	9	2	6	2	∘
610	4	4	2	0	∘	19	2	13	2	•
611	24	4	23	4	•	2	1	2	0	∘
612	16	7	29	4	•	7	1	6	1	∘
613	17	3	15	3	•	12	1	13	1	∘
614	6	11	9	1	∘	17	1	23	2	•
615	61	11	36	5	•	0	0	0	0	∘
616	85	8	43	6	•	0	0	0	1	∘
617	61	6	26	4	•	0	0	0	1	∘
618	10	1	17	3	•	0	0	0	2	∘
619	1	2	10	2	•	0	0	0	2	∘
620	0	0	0	0	•	4	1	2	3	∘

Table 1: Tabular illustration of the small intersection policy. The column marked *wt* is the sum wait time, and n denotes the number of cars waiting. Green lights are marked with a $\bullet$, red lights with a $\circ$. These are the settings that were chosen as a function of the situation shown in the same row. This policy is taken from the same run as displayed in Fig. 6.

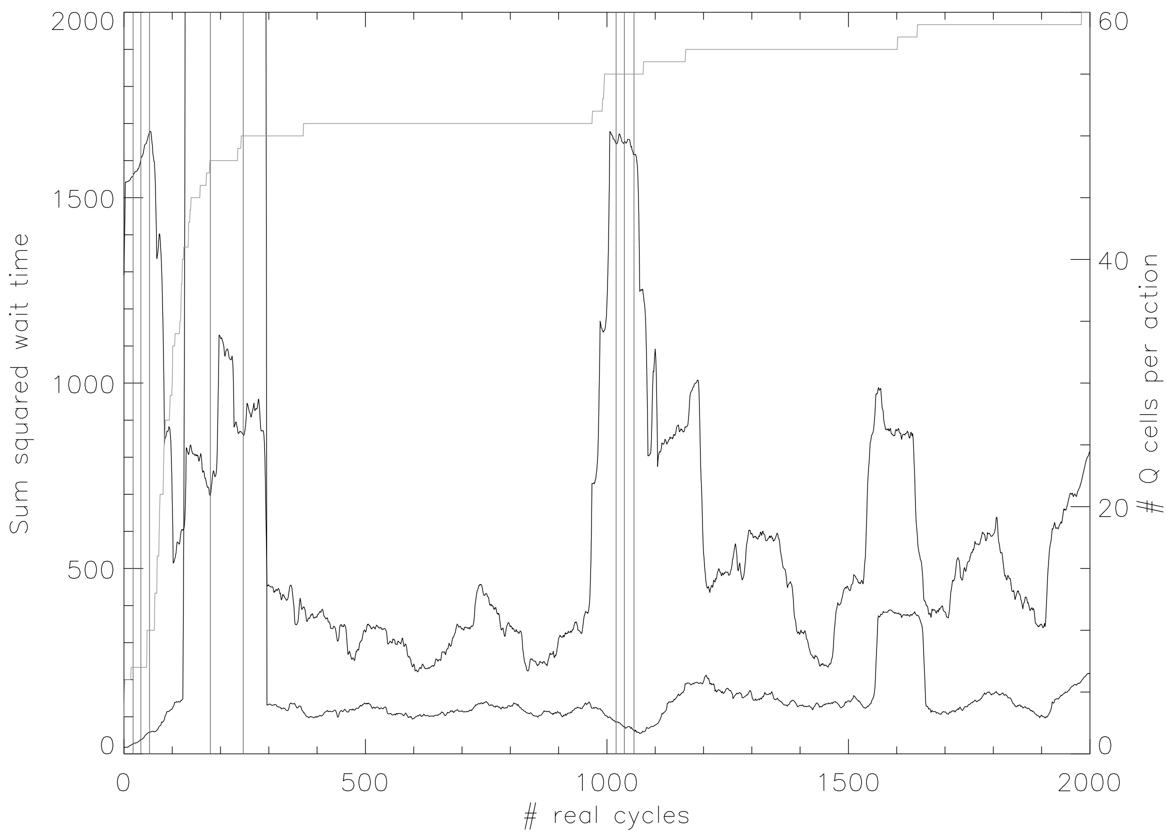

Figure 6: Sample behavior of the small intersection. At iteration 0, the arrival rates of the simulated intersection were 0.1/sec and 0.6/sec, respectively. At iteration 1000, these rates were reversed. The model rates were initialized to 0.1/sec and allowed to adapt. The top solid graph shows the real summed squared waiting times, the bottom one the model ones. Both are box-car smoothed over an interval of 99 points. The vertical lines indicate where a new trial was started because the squared wait time of the environment exceeded 10000. The light graph shows the number of Q table entries generated.

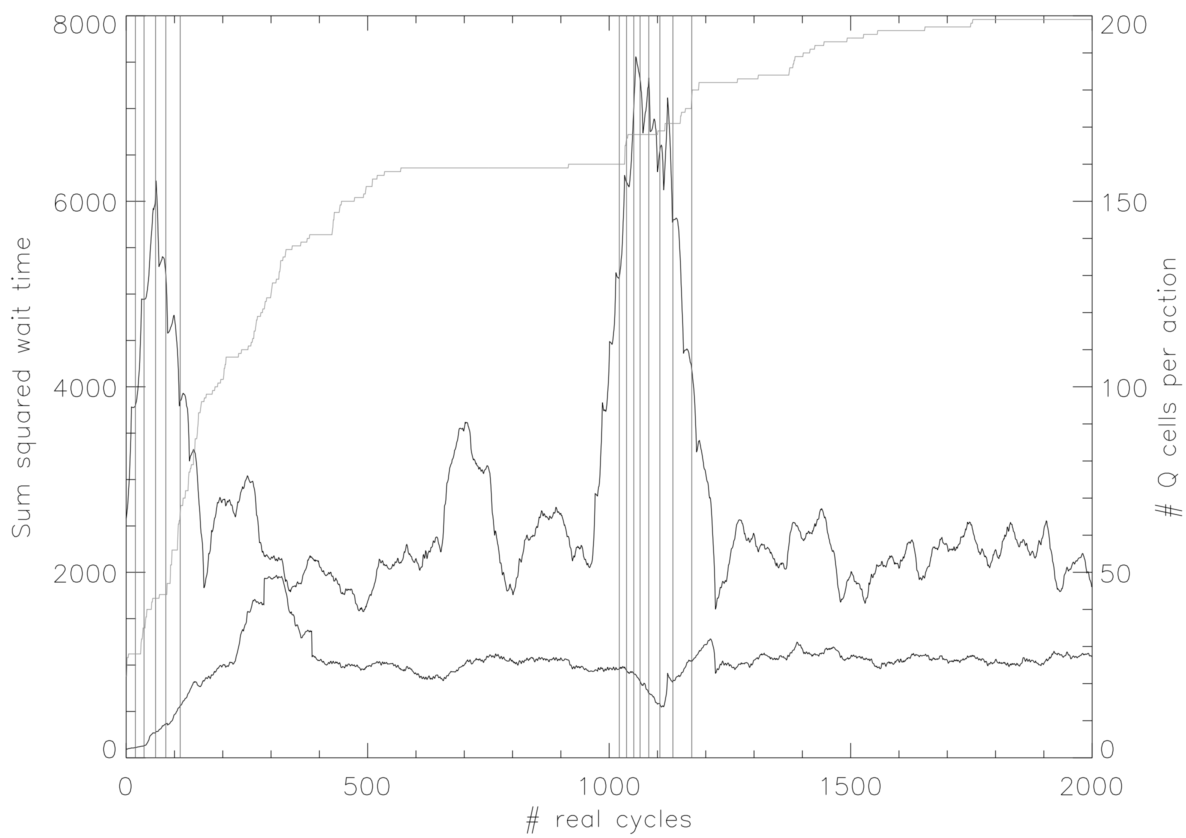

Figure 7: Sample behavior of the medium intersection. At iteration 0, the arrival rates were 0.4 at both lanes coming from the south, and 0.1 for all other lanes. At iteration 1000, they were reset to 0.1, and the two lanes going straight across were set to 0.4. Model parameters had to adapt as before. The graphs are smoothed as in Fig. 6. The squared wait time limit was 20000.

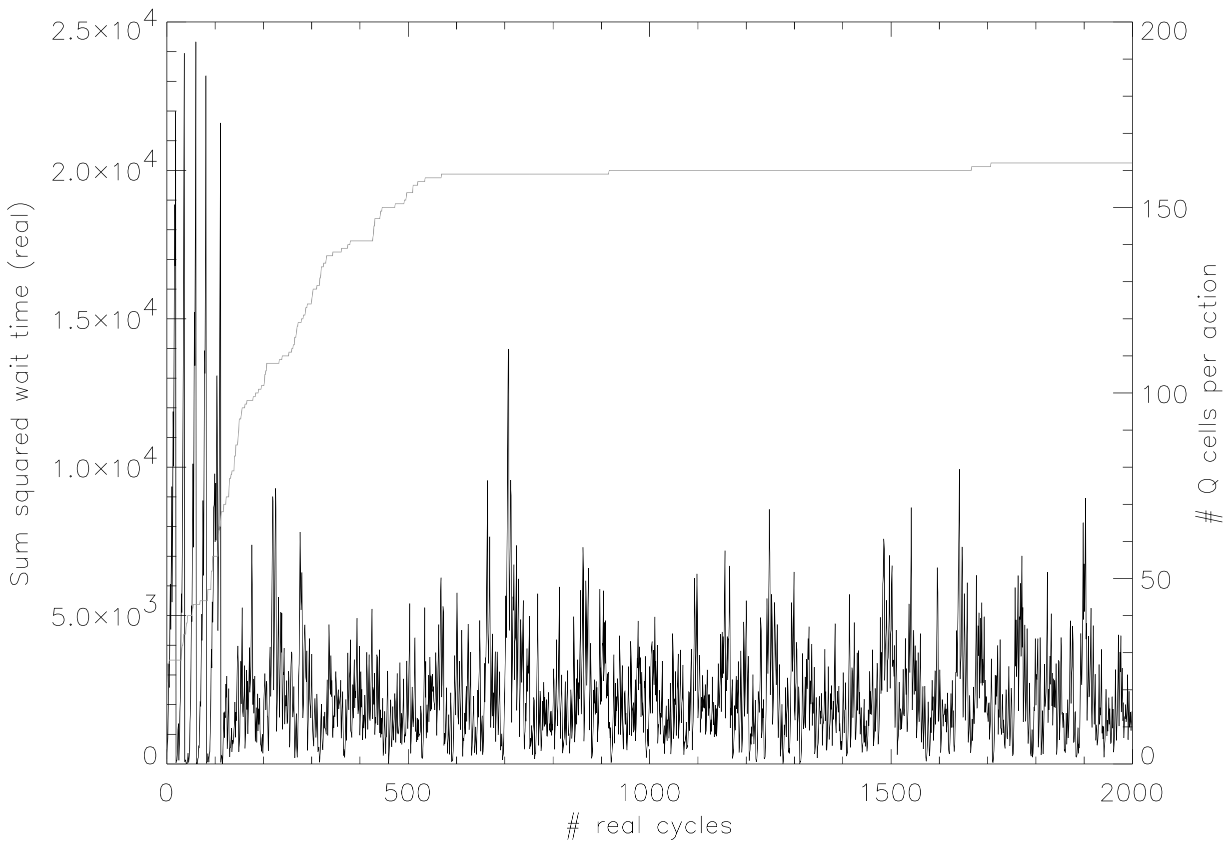

Figure 8: Medium intersection (unsmoothed). Initial arrival rates at the real intersection are as in Fig. 7, but are left unchanged. Only the real squared waiting time is shown. The simulated intersection was reset at each of the five initial peaks.

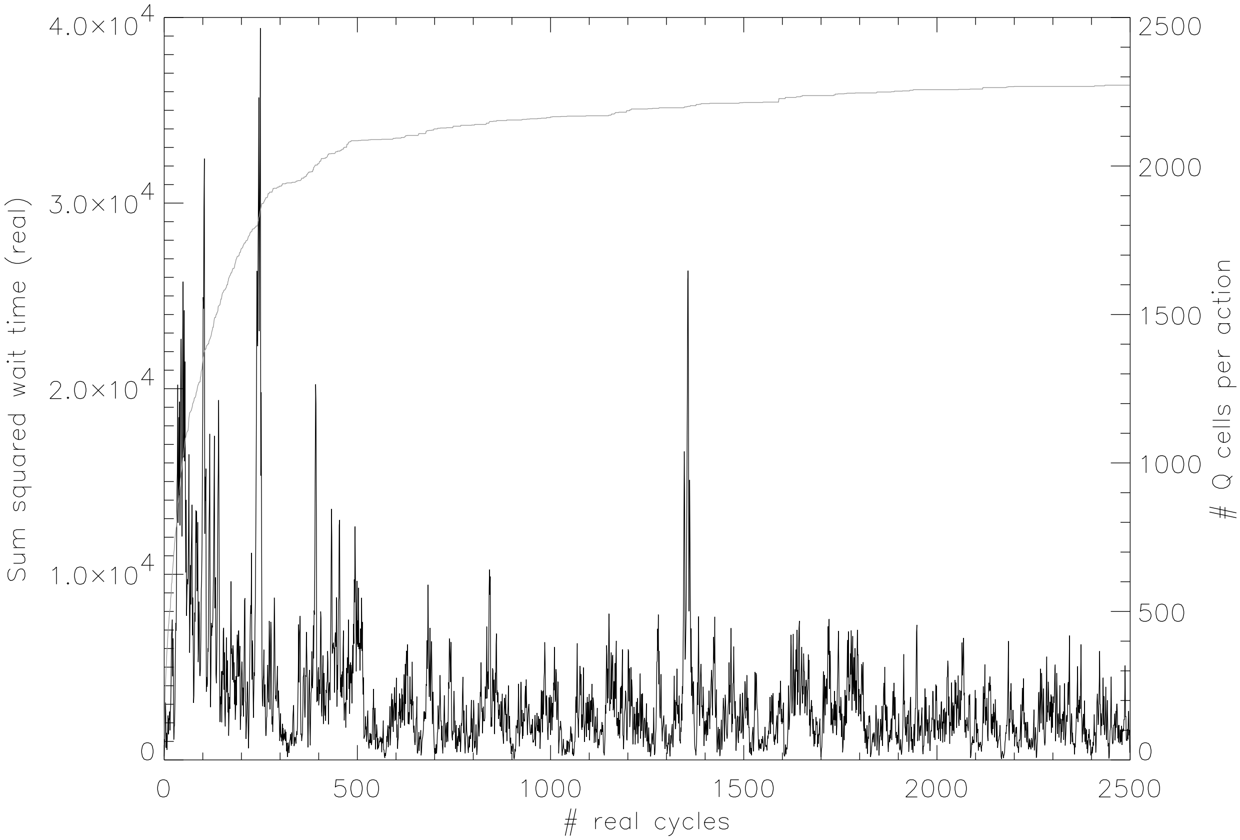

Figure 9: Large intersection with fixed model parameters that match the simulator. The arrival frequencies for the two lanes going straight from north to south and vice versa were set to 0.2/second, and to 0.08/second on all other lanes. The system was never reset.

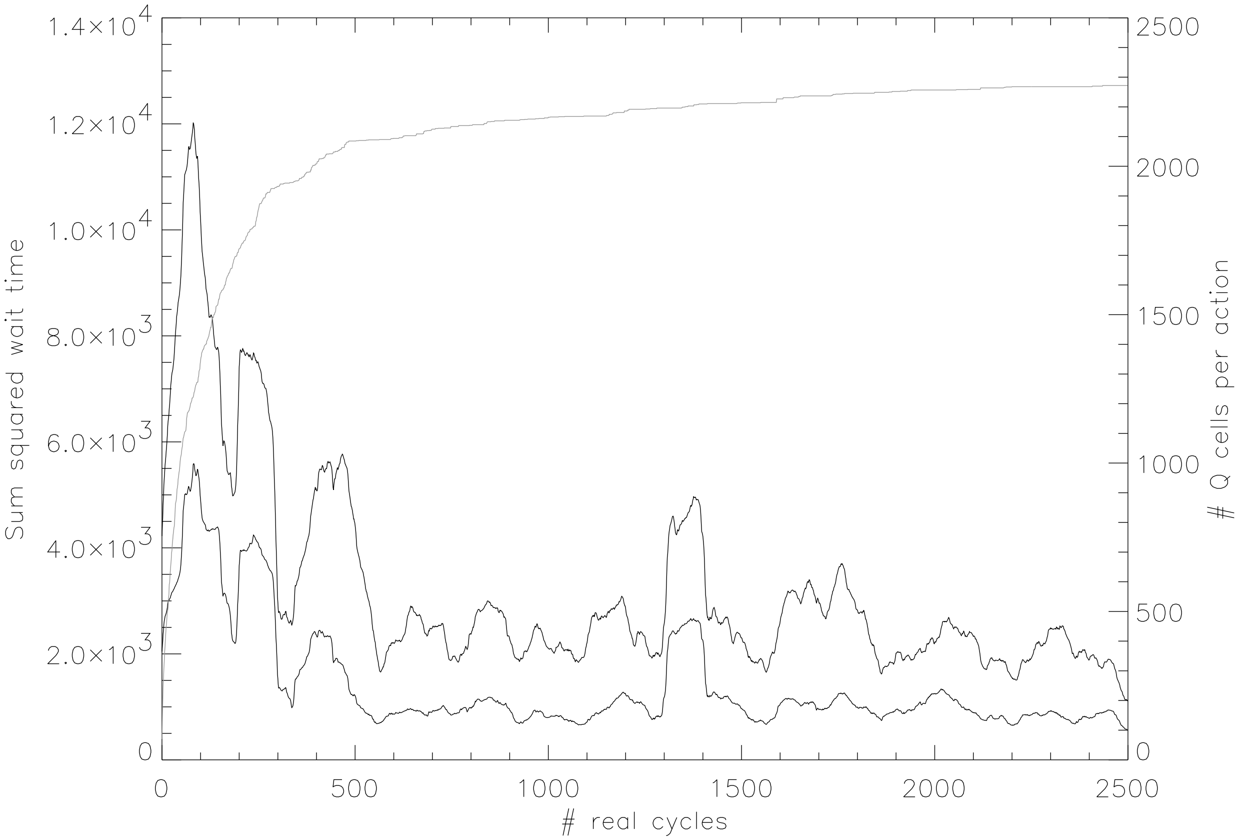

Figure 10: Large intersection like Fig. 9, but data are boxcar smoothed. Both simulated (top) and model (bottom) squared waiting times are shown.

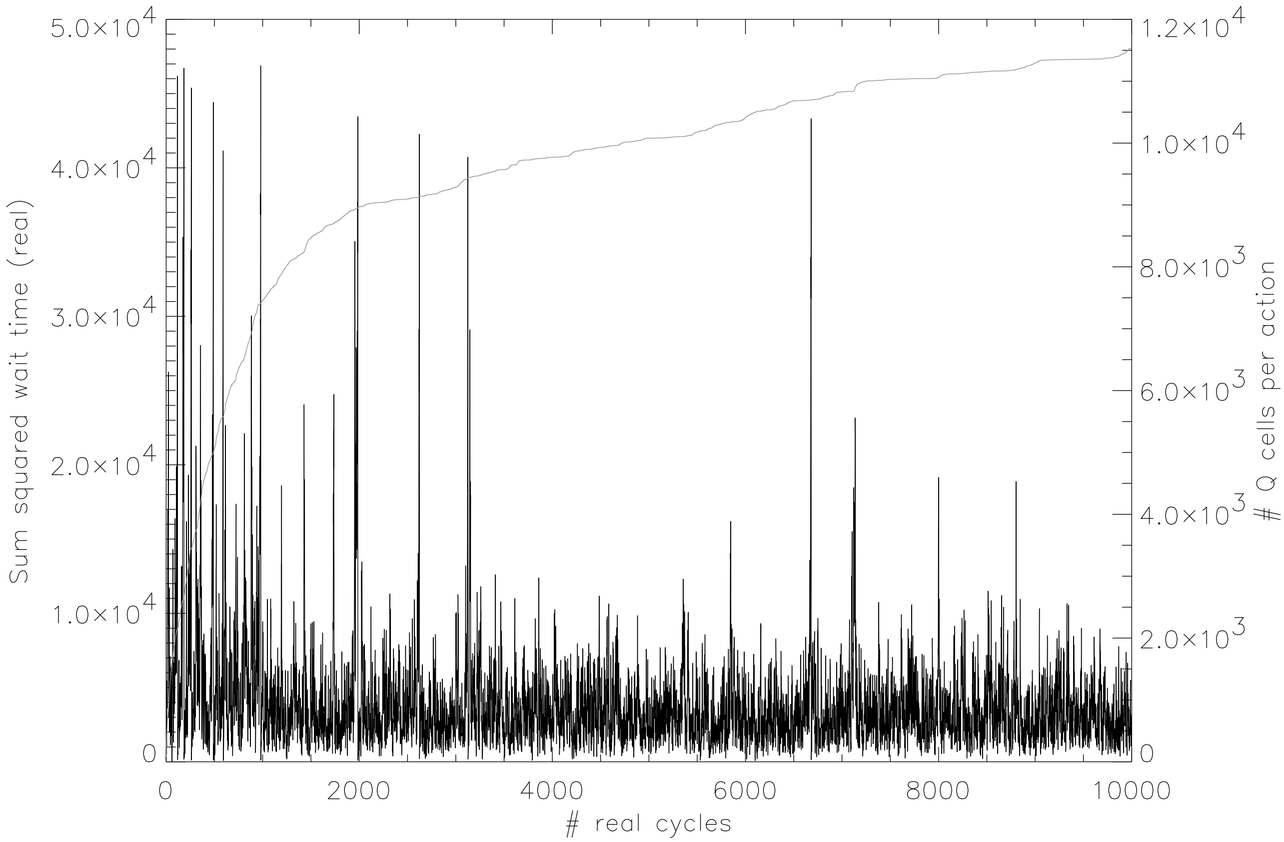

Figure 11: Large intersection. The arrival rates of the simulated intersection were set as before, but the model rates had to adapt from their initial default value of 0.1/sec. The sum square wait time threshold was set at 40000. The simulated intersection was reset at each of the ten peaks exceeding this threshold (at cycles 119, 185, 263, 490, 592, 981, 1986, 2624, 3127, and 6681).

6 Results

6.1 Policy learning

In the following, the policy is evaluated with respect to the model only to avoid the influence of the discrepancy between model and environment.

After every ith model action, the policy is evaluated as follows. A copy of the model is initialized with zero cars. Learning is completely turned off. Then, the agent runs one trial of one hundred observations and actions on the model. This is done for a given number of model actions to complete a run. Several of such runs are then averaged.

Performance is measured in terms of the sum of

- the sum of the squared waiting times of the cars passed, and
- the remaining sum squared waiting time at the end of the trial.

Thus, the accuracy of the model is not tested; neither is the efficacy of the model update algorithm. However, the learned policy is in fact influenced by the environment because it determines when the system is reset (cf. Sec. 5.5). Since all test trials start off with an empty intersection, but even a well-trained action value function provides bad estimates for this case, outliers are likely to occur during testing.

Figures 12 and 13 show the learning curves for the small and medium intersections. These were averaged across several independent runs. For the small intersection, curves containing extreme outliers (by a factor between 10 and 10^5, only after cycle 100) were discarded.

The medium intersection did not produce such extreme outliers, but they are still visible in the graph. They appear to occur less frequently with growing number of cycles. Waiting times are generally higher here because cars arrive at six lanes rather than two.

Both averaged learning curves converge to some minimum as the Q cell curve flattens out. In fact, none of the separate runs achieved significantly better performance at any time. For the small intersection, this asymptote is so low that it can be taken as near the optimum. In the medium intersection, this is not quite so obvious. Unfortunately, verification of optimality is non-trivial in this context.

It should also be noted that it is generally not at all obvious what the maximum feasible arrival rates are. An appropriate, but time-consuming way would be to run simulator interactively and set the lights manually for a number of cycles. However, this comparison would not be quite fair because one would make the decisions based on the actual waiting cars rather than some functional approximation.

6.2 Other policies

For comparison, a random policy was run on the simulated intersection. Not surprisingly, it did well as long as all car arrival frequencies were equally low. However, when the paramters as described above were used, the waiting times steadily increased, and the system had to be reset over and over again.

No heuristic policies were implemented since they seemed too complex considering the limited time (except for the small intersection where a good policy is rather obvious).

7 Discussion

7.1 The Model

In principle, it is possible to build an open loop control system that uses very sparse information to update its model. At the same time this is its major limitation: A better model would probably greatly improve the performance of the system, but the amount of work necessary to design a "good" model seems inappropriate for an RL system that is supposed to *learn* a good policy. It proved to be very hard to develop this model, although it is still far from perfect. In fact, a lot more time was spent on model improvement than on the development of the reinforcement learner.

A better approach would rely on situation descriptions which are immediately available or can be derived from observations of the environment, thus applying closed loop control. Very little more sophisticated sources are required to obtain much better information, e.g. two or more inductive loops per lane that can be used to estimate the length of a queue or the speed of a passing car, or can tell in advance that a car is coming on a low-traffic road, thus enabling the controller to set its lane to green before the car has actually arrived at the crossing.

A goal of this work has been to investigate whether a policy could adapt to a changing environment in real time. This seems not to be the case for nontrivial road intersections. Hence, information about the traffic densities should be included in the situation description.

However, this leads to a huge state space which requires a much more sophisticated function representation (cf. Sec. 7.2). A considerable amount of off-line learning on a simulator seems to be essential for a real application.

7.2 Function Representation

The present implementation uses a discretized non-overlapping tabular state space. Learning should be greatly improved by using generalization. For instance, a lane with a maximal sum of waiting times should preferably be set to green, no matter what the actual sum is. This suggests Generalized Grid or CMAC coding.

Since with some careful thought dimensions suitable for generalization can be specified, I suggest that a neural network be used only if its architecture is specially designed to support this and discourage unwanted generalization.

A Parameters

A.1 Agent

duration of observation	5 sec
# model cycles between real actions	100
probability of random model action	0.2
discount factor γ	0.7
step size α	0.1

A.2 Lane

general parameters:	
car arrival frequency f	(variable)
time needed per car to pass	1 second
time yellow	2 seconds
additional simulator parameter:	
standard deviation of arrival time	$1/f$
additional model parameters:	
α_f (see Fig. 2)	0.1
β_f (see Fig. 2)	1.01

A.3 Q function table

For the small intersection, the waiting time per lane was discretized into 8 buckets at the following thresholds:

$$10, 20, 50, 100, 500, 2000, 10000$$

For the medium and large intersections, 4 buckets were used with boundaries

$$50, 200, 1000$$

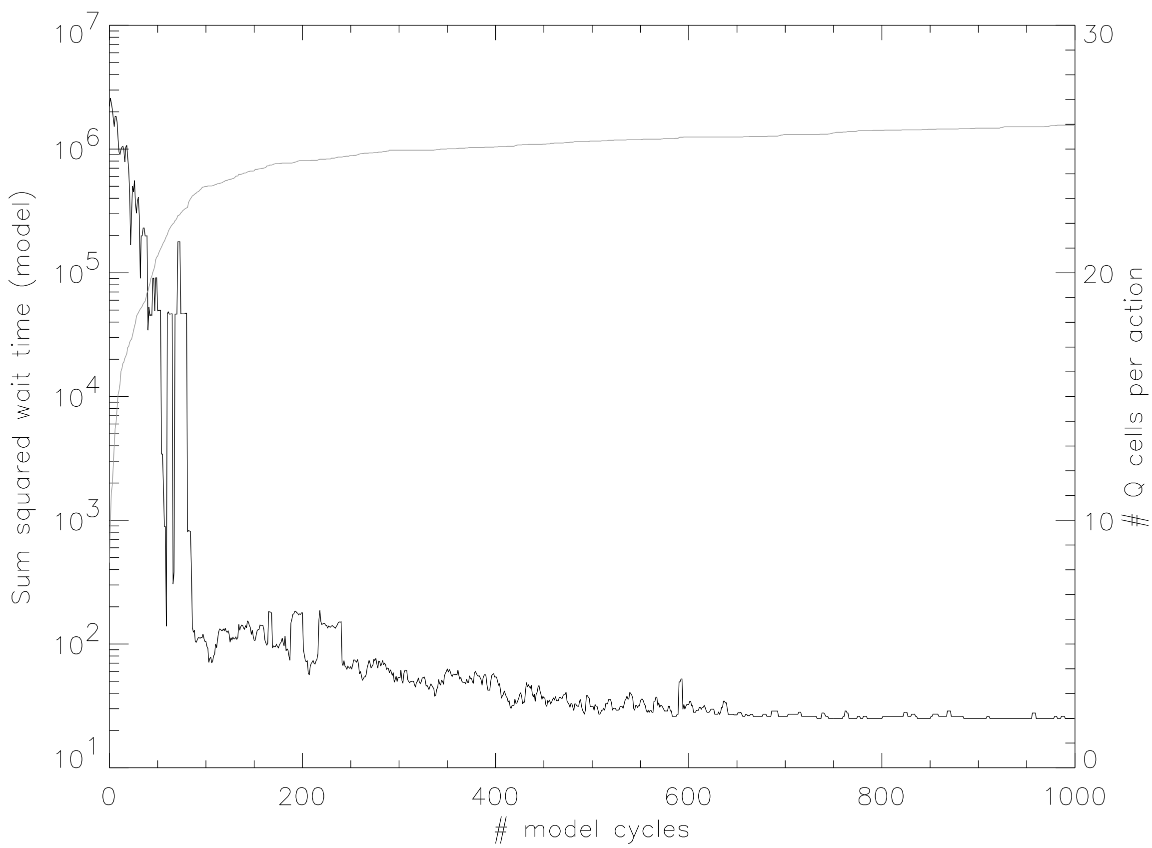

Figure 12: Learning curve for the small intersection. The model was evaluated after every model cycle. 90 runs were averaged, and 10 more curves with apparent outliers were discarded. A few such outliers are still visible (in the area between 50 and 100 model cycles). The light curve shows the number of Q function cells generated. A real action was taken after every 100th model update.

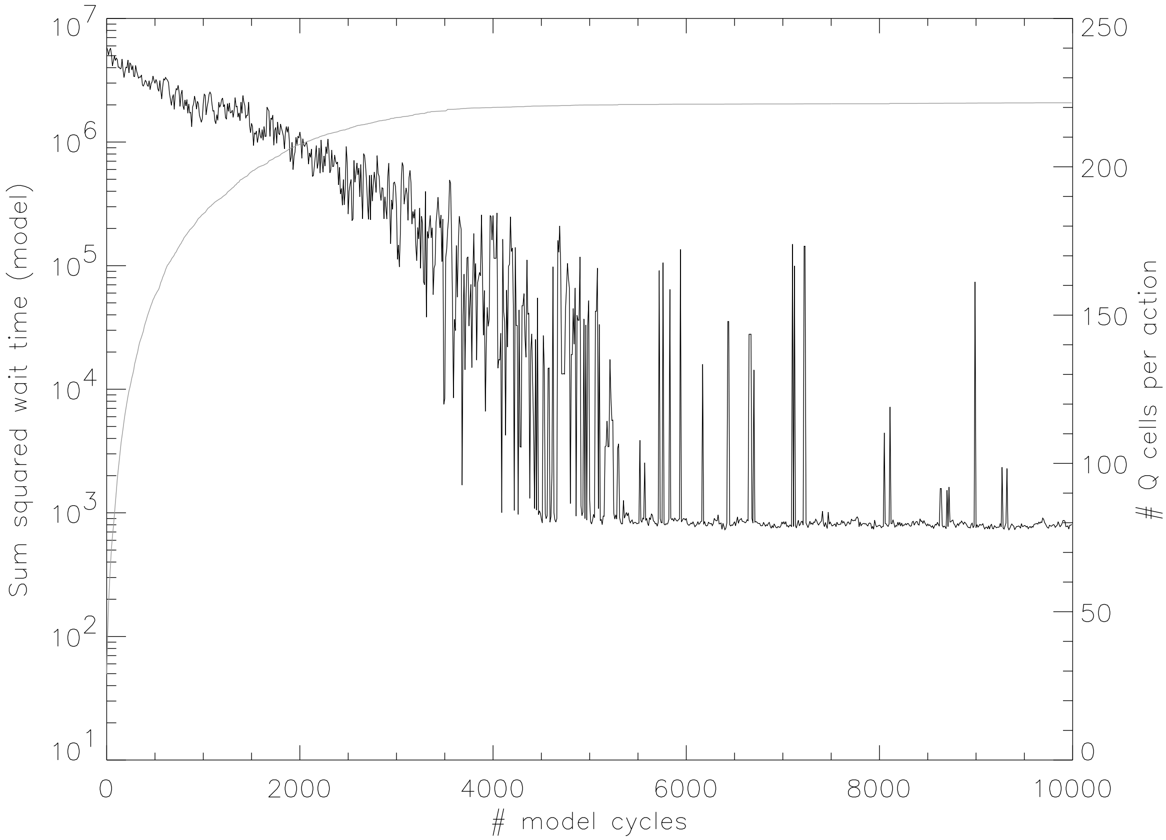

Figure 13: Learning curve for the medium intersection. The model was evaluated after every 10th model cycle. 100 runs were averaged.