Traffic Light Scheduling using Policy-Gradient Reinforcement Learning

Silvia Richter
Griffith University & NICTA *
Brisbane, Australia
silvia.richter@nicta.com.au

Abstract

Most currently deployed traffic light controllers use hand tuned policies with limited automatic adaption, while state-of-the-art research controllers assume unrealistically good models of the traffic. We use a policy-gradient reinforcement learning approach to traffic light optimization which maps sensor observations directly to control signals. Our approach shows promising results in a number of simulated traffic scenarios, without requiring explicit traffic models.

Introduction

Optimizing the performance of existing road networks is a cheap way to reduce the environmental, social, and financial impact of ever increasing volumes of traffic. In this work, we aim to optimize the scheduling of traffic lights by trying to find, for every traffic light in a given road network, the optimal duration and order for each of the different possible phases of the light.

Traffic light optimization can be naturally cast as a reinforcement learning (RL) problem. Unfortunately, it is a very hard problem for several reasons: it has a continuous state space and infinite horizon, is only partially observable and difficult to model. We employ policy-gradient methods, that perform search in policy-space via gradient-ascent, as they allow for local convergence under function approximation and partial observability. In particular, we focus on two different algorithms: the recent natural actor-critic (NAC) algorithm (Peters, Vijayakumar, & Schaal 2005), and a simple online policy-gradient (PG) approach for comparison.

This work has grown out of an interaction with the Sydney Road Traffic Authority, which is striving to improve their traffic control system SCATS. Our choice of controls, observations, and algorithms all aim for practical large-scale traffic control. Although our results are based on a simplified traffic simulation system, we could theoretically attach our learning system to real-world traffic networks. In our experiments, we compare against a SCATS-inspired baseline.

Background

The optimization problem consists of finding signalling schedules for all intersections in the system that minimize the average travel time, or similar objectives. This is complicated by the fact that many of the influencing state variables cannot be readily measured. Most signal controllers in use today rely only on state information gained from *inductive loops* in the streets.

Existing approaches to Traffic Control can be grouped into three categories. *Fixed time* control strategies are calculated off-line, based on historical data. *Traffic responsive* strategies are real-time, calculating their policies from car counts determined from inductive-loop detectors. SCATS is one such system in use around the world. *Third generation* methods employ sophisticated dynamic traffic models and try to find optimal durations for all phases given a fixed *phase scheme*, e.g. by dynamic programming (Papageorgiou 1999). Reinforcement learning has also been applied (Wiering 2000), but in a way that uses a value function for each car, which is impractical in today's world. Common to most approaches is that they deal with the insufficient state information by maintaining a *model* of the traffic situation, derived from available sensor counts. However, imperfections in the model is a source of errors and performance may consequently suffer. Our methods avoid modelling.

Partially Observable MDP Formulation

We cast the traffic problem as a partially observable Markov decision process (POMDP) with states s in a continuous space $\mathbb{S}$. The system is controlled by stochastic actions $\mathbf{a}_t \in \mathbb{A}$ drawn from a random variable (RV) conditioned on the current policy parameters θ_t, and an observation of the current state $\mathbf{o}(s_t)$, according to $\Pr(a_t|\mathbf{o}(s_t), \theta_t)$. In our setting the observation function $\mathbf{o}(s_t)$ is deterministic. The state is an RV evolving as a function of the previous state and action according to $\Pr(s_{t+1}|s_t, \mathbf{a}_t)$.

A common objective function in traffic control is the average vehicle travel time. However, it is impossible to identify a particular car in the system, let alone what its travel time was. We use a measure that can be directly obtained from existing sensors: we treat each intersection as a local MDP and count all cars that enter the intersection with loop detectors. The instant reward $r_{t,i}$ in time step t is then the number of cars that entered intersection i during t. The objective for each intersection i is to maximize the normalized *discounted*

*NICTA is funded by the Australian Government, in part through the Australian Research Council.

throughput:

$$R_i(\theta) = \mathbb{E}\left\{ (1-\gamma) \sum_{t=0}^{\infty} \gamma^t r_{t,i} \middle| \theta \right\}.$$

Here, discounting (by γ) is important because it ensures that the controller prefers to let cars pass through as *early* as possible. The use of *local rewards*, as opposed to one global reward for the entire system, speeds up learning dramatically, reducing the worst case sample complexity by an order of magnitude (Bagnell & Ng 2006). Unfortunately, the value of $R_i(\theta)$ depends directly on the *local steady state distribution* $\Pr(s_i|\theta)$. Thus changes to the policy of neighbouring intersections can adversely impact intersection i, by influencing the distribution of s_i. A sufficiently small *learning rate* allows controllers to adapt to this effectively non-stationary component of the local MDP. We may fail to find the globally optimal cooperative policy, but it has proven very effective empirically.

Policy Gradient Ascent

Searching policy space is of interest in RL because it can be easier to learn policies directly than to estimate the values of all states. Policy gradient (PG) ascent methods do this by performing gradient ascent on the parameters of a parametrized policy function. They are guaranteed to converge to a *local* optimum of their optimization function under appropriate conditions. In combination with Monte-Carlo-like exploration of the environment, PG methods allow efficient handling of large state spaces.While PG methods offer only local convergence guarantees, they do not suffer from the convergence problems exhibited by pure value-based methods under function approximation or partial observability (Sutton *et al.* 2000). On the other hand, PG methods have suffered from slow convergence compared to value methods due to high variance in the gradient estimates. The natural actor-critic method (NAC) (Peters, Vijayakumar, & Schaal 2005) improves this with a combination of PG methods, natural gradients, value estimation, and least-squares temporal-difference Q-learning (LSTD-Q). The original NAC computes gradient estimates in a batch fashion, followed by a search for the best step size. We have developed an online version of NAC (Richter, Aberdeen, & Yu 2007), as we cannot perform a line search on a real world traffic system because *during* the line search we may try arbitrarily poor step size values. Furthermore, the gradient estimates in our POMDP are noisy, disadvantaging batch methods (Bottou & Le Cun 2004).

Our Simulator

We implemented a simple traffic simulation system, aiming at high simulation speed rather than at an accurate model of traffic flow. However, we modelled the phase control protocol and sensor system based on information from the Sydney Traffic Authority. Given that the learning algorithm does not depend directly on a model of the system, just the ability to interact with it, our controller can be plugged into a more accurate simulation without modification. Our simplifying assumptions include: all vehicles move at uniform speed; road length is a multiple of the distance cars travel in one step; we ignore interactions between cars or the relative positions of cars within one road segment except in intersection queues. For further details see Richter, Aberdeen and Yu (2007).

The Control Architecture

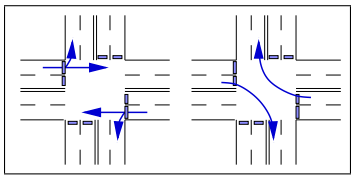

Fig. 1: The intersection model, showing 2 phases and detectors.

Commonly, Australian intersections have 2 to 6 phases. Ours have 4 phases: for traffic coming from east or west (EW) straight and left turns, EW right turns, north/south (NS) straight and left turns, and NS right turns (see Fig. 1). At each time step (corresponding to about 5 seconds real-time) the controller decides which phase to activate in the next step. We do not restrict the order of phases, but to ensure a reasonable policy we enforce the constraint that all phases must be activated at least once within 16 time steps.

The controller input for intersection i is $\mathbf{o}_{t,i}$, constructed as follows: **Cycle duration**: 16 bits, where the nth bit is on in the nth step of the cycle, supporting time based decisions. **Current phase**: 4 bits, indicating the previous phase. **Current phase duration**: 5 bits, indicating that we have spent no more than 1, 2, 4, 8 or 13 continuous time steps in the current phase. **Phase durations**: 5 bits per phase, in the same format as current duration, counting the total time spent in each phase in the current cycle. **Detector active**: 8 bits for the 8 loop sensors indicating whether a car is waiting. **Detector history**: 3 bits per loop sensor, indicating a saturation level of more than 0, more than half capacity, or capacity, in the current cycle. **Neighbour information**: 2 bits, giving a delayed comparison of the flows from neighbouring intersections, indicating where traffic is expected from.

Following an approach similar to the (non-temporal version of) the FPG planner (Aberdeen & Buffet 2007), our controller uses a linear approximator to map observations $\mathbf{o}_{t,i}$ for intersection i to output values corresponding to the 4 phases. These values are turned into a probability *distribution* $\tilde{\mathbf{a}}_{t,i}$ over the 4 phases using the soft-max function.

Experiments

We tested the performance of online NAC against the simple online PG algorithm OLPOMDP (Baxter, Bartlett, & Weaver 2001), and also against two baseline controllers: (1) a uniform controller giving equal duration to all phases; (2) A SCATS inspired adaptive controller called SAT that tries to achieve a saturation of 90% (thought to be used by SCATS) for all phases. The exact details of SCATS are not available. We aimed to recreate just the *adaptive* parts of SCATS, no hand-tuned elements. We conducted 5 experiments, which for space reasons are only described very briefly here.

Fluctuating: we focus on an intersection in the centre of a crossroads. The traffic volume entering the system on the NS and EW traffic axes is proportional to a sine and cosine function of the time, respectively. Thus the optimal policy at the centre intersection also oscillates with the traffic volume.

Tab. 1: Comparison of travel times (TT) for all methods and all scenarios. Quoted for the PG algorithms are best results achieved within a certain max. run time (long enough to allow reaching a quasi-steady state).

Scenario	Random	Unif.	SAT	NAC	OLPOMDP
Fluct	250.0	102.0	21.5	14.3	13.4
Burst	197.0	35.0	18.4	13.4	13.5
Offset	17.9	15.0	12.0	8.0	8.0
A.D.	251.0	74.2	17.2	15.8	16.0
100 int.	60.5	54.7	35.1	29.8	27.9

Tab. 2: Optimization run times for all scenarios for the PG algorithms. Optimization was performed for t steps. 'Secs' is wall-clock time.

Scenario	NAC		OLPOMDP	
	t	secs	t	secs
Fluct.	$4.5 \cdot 10^6$	860,549	$1.1 \cdot 10^9$	491,298
Burst	$4.4 \cdot 10^6$	25,454	$9.7 \cdot 10^8$	35,572
Offset	$2.1 \cdot 10^6$	1,973	$6.3 \cdot 10^8$	8,546
A.D.	$9.3 \cdot 10^7$	867,267	$2.2 \cdot 10^9$	807,496
100 int.	$2.9 \cdot 10^5$	1,077,151	$3.0 \cdot 10^8$	1,029,428

This scenario is realistic because upstream intersections release periodic bursts of traffic, which then disperse as they travel along the road. SCATS is known to adapt too slowly to deal well with such situations. On average 3 cars enter the system per time step from each direction, each car needs to travel 12 space units (hence, minimum travel time is 12).

Our results quote the average travel time (TT). Tab. 1 shows that NAC and OLPOMDP both improve upon the uniform controller and SAT. The two PG algorithms get similar results across all scenarios. However, Tab. 2 shows that NAC does so in up to 3 orders of magnitude fewer learning steps, but sometimes requires more CPU time. In a real deployment NAC would be able to keep up with real-time, thus we are much more concerned about reducing learning steps. The tables quote a single run with tuned parameters. To check the reliability of convergence and compare the properties of the two algorithms, Fig. 2 displays the results of 30 runs for both algorithms in one of our scenarios.

Burst: intersection controllers can learn to cooperate by using common observations. We make use of *only* the neighbours feature in the observations, so the controller must use the detector counts of its neighbours to anticipate traffic.

Offset: demonstrates learning an offset (green wave) between neighbouring intersections on a main road, a feature that needs to be hand-tuned in SCATS. Both PG methods learned an optimal policy. SAT performed badly because it had no means of implementing an offset.

Adaptive Driver: a scenario designed to require *global* optimization. We use the number of cars in the system as the global reward (which minimizes the average travel time assuming constant input of cars). This reward is hard to estimate in the real world, but we want to demonstrate the ability of the system to learn cooperative policies using a global reward. The policies learned by the PG approaches were radically different from SAT's. A slightly larger volume of

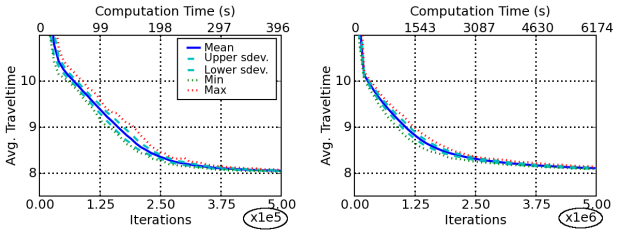

Fig. 2: Convergence properties of NAC (left) compared to OLPOMDP (right) over 30 runs in the Offset scenario.

of vehicles made SAT cause permanent traffic jams, while the PG algorithms still found the correct policy.

Large Scale Optimization: a 10×10 intersection network with randomly chosen routes for cars. We used local rewards and all observations. OLPOMDP gave an average travel time improvement of 20% over SAT even though this scenario was not tailored for our controller. Such savings in a real city would be more than significant.

Conclusion

We employed online stochastic policy-gradient procedures for a distributed road traffic problem to demonstrate where machine learning can improve upon existing traffic controllers. Our approach yields good results while obviating the need for a model. In future work we will use realistic simulators and develop improved algorithms to cope with the increased noise and temporal credit assignment problem.

References

Aberdeen, D., and Buffet, O. 2007. Temporal probabilistic planning with policy-gradients. In *Proc. ICAPS 2007*. To appear.

Bagnell, J. A., and Ng, A. Y. 2006. On local rewards and scaling distributed reinforcement learning. In *Proc. NIPS 2005*, volume 18.

Baxter, J.; Bartlett, P.; and Weaver, L. 2001. Experiments with infinite-horizon, policy-gradient estimation. *JAIR* 15:351–381.

Bottou, L., and Le Cun, Y. 2004. Large scale online learning. In *Proc. NIPS 2003*, volume 16.

Papageorgiou, M. 1999. *Traffic Control. In Handbook of Transportation Science*. R. W. Hall, Editor, Kluwer Academic Publishers, Boston.

Peters, J.; Vijayakumar, S.; and Schaal, S. 2005. Natural actor-critic. In *Proc. ECML 2005*, 280–291.

Richter, S.; Aberdeen, D.; and Yu, J. 2007. Natural actor-critic for road traffic optimisation. In *Proc. NIPS 2006*, volume 19.

Sutton, R. S.; McAllester, D.; Singh, S.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Proc. NIPS 1999*, volume 12.

Wiering, M. 2000. Multi-agent reinforcement learning for traffic light control. In *Proc. ICML 2000*.