Transportation Research Part C 17 (2009) 456–474

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Adaptive traffic signal control using approximate dynamic programming

Chen Cai *, Chi Kwong Wong, Benjamin G. Heydecker

Centre for Transport Studies, University College London, London WC1E 6BT, United Kingdom

ARTICLE INFO

Article history:
Received 6 May 2008
Received in revised form 31 January 2009
Accepted 14 April 2009

Keywords:
Traffic signal
Dynamic programming
Approximation
Adaptive
Reinforcement learning

ABSTRACT

This paper presents a study on an adaptive traffic signal controller for real-time operation. The controller aims for three operational objectives: dynamic allocation of green time, automatic adjustment to control parameters, and fast revision of signal plans. The control algorithm is built on approximate dynamic programming (ADP). This approach substantially reduces computational burden by using an approximation to the value function of the dynamic programming and reinforcement learning to update the approximation. We investigate temporal-difference learning and perturbation learning as specific learning techniques for the ADP approach. We find in computer simulation that the ADP controllers achieve substantial reduction in vehicle delays in comparison with optimised fixed-time plans. Our results show that substantial benefits can be gained by increasing the frequency at which the signal plans are revised, which can be achieved conveniently using the ADP approach.

1. Introduction

Operating traffic signals in urban areas requires proper timings, so that varying demands can be managed effectively. Conventional algorithms, which are optimised off-line, usually generate a library of signal timing plans each with fixed stage duration and sequence. Plans are retrieved from the library for implementation according to the time of day and the day of week. Such plans require manual maintenance and updating, otherwise the performance declines at a rate of about 3% per year (Bell and Bretherton, 1986). Most of the operating signal systems today are traffic responsive (or vehicle actuated). The responsiveness to traffic is that the allocation of green times is adjusted according to real-time traffic information. The real-time traffic data are usually detected by using inductive loops. While bringing substantial benefits in reducing vehicle delays and stops, responsive systems are usually constrained by preset control parameters, such as cycle length and stage sequence. Ideally, for optimisation over time, the controller should operate signals regardless of cycle and stage constraints. This requires a dynamic controller to act according to detected traffic and update its control parameters online without human intervention, thus being adaptive.

Dynamic programming (DP) developed by Bellman (1957) is so far the only exact solution for optimisation over time. It decomposes a control problem to a series of sub-problems which we denote as *step*, which corresponds to discrete segments of time in real-time control problem. Associated with each step is a set of *state* variables that give information on the controller and the traffic environment at that time. The DP recursively calculates *Bellman's equation* backwards step-by-step to find the optimal action that transfers the system from the current state to a new state. In doing this, the DP generates backwards in time a sequence of optimal actions that guarantee global optimality. The DP solutions to traffic signal control are studied in Robertson and Bretherton (1974) and Gartner (1983). The results show that using DP can reduce about 56% vehicle delays from the best fixed-time plans. Nevertheless, the DP's implication for real-time traffic signal control is limited. The

* Corresponding author. Tel.: +44 (0)20 76790467; fax: +44 (0)20 76791567.
E-mail addresses: c.cai@ucl.ac.uk (C. Cai), ckwong@transport.ucl.ac.uk (C.K. Wong), ben@transport.ucl.ac.uk (B.G. Heydecker).

0968-090X/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.trc.2009.04.005

computational demand in the recursive calculation of Bellman's equation is exponential to the size of the state space, the information space and the action space. This scenario is often described as the 'Three Curses of Dimensionality' (Powell, 2007). Furthermore, the DP requires a complete information on the time period in which the controller seeks optimisation. In real-time operation, however, traffic detectors may supply only 5–10 s data of future arriving vehicles.

To overcome the difficulties in applying DP and to preserve the fundamental features of dynamic control, a favourable option is approximation. An approximation to DP usually aims to reduce state space by replacing a look-up table of state values with aggregations or a continuous approximation function. Such an approach is frequently denoted as *approximate dynamic programming* (ADP). In this paper, we limit the study to continuous approximation function only. Since we may not know the appropriate values of functional parameters *a priori*, it is preferable that the controller acquires adaptive features that update the parameters according to both the changes in the prevailing traffic and the observation of the controller's interaction with the traffic environment. An approach that applies the fundamentals of dynamic programming to learn from interactions with the environment is *reinforcement learning*. This approach uses the dynamic programming formula to map the system state to action. The action changes the environment, and this change is communicated back to the controller through a scalar reinforcement signal. The functional parameters are updated by specific learning techniques upon receiving the reinforcement signal. In this study, we investigate two learning techniques, *temporal-difference* and *perturbation learning*.

In this paper, we show that an adaptive traffic signal controller using ADP and reinforcement learning is capable of reducing vehicle delays substantially from the best fixed-time control, while being computationally efficient. The numerical experiments presented here are limited to an isolated intersection with multiple signal stages, but the implication extends to distributed control in traffic networks.

This paper is organised as follows. In Section 2, we review the existing traffic signal control systems, from which we identify the scope for development and set objectives for the ADP controller. In Section 3, we introduce the fundamentals of the ADP, reinforcement learning and the specific learning techniques, based on which the control algorithms are formulated for traffic signals. Section 4 contains numerical experiments and results. Section 5, contains conclusion of this study and the scope for future research.

2. Traffic signal control

The operation of traffic signal settings can be broadly classified into off-line and online approaches. Off-line methods use historical traffic data as inputs for the calculation of signal plans. By contrast, online approaches use traffic information collected in real-time from on-street detectors to develop responsive signal control. Here, we limit the review in online control systems. Among online approaches, those that collect real-time traffic information from detectors and use it to calculate up-to-date signal settings for implementation pertain to *responsive control*; those that use real-time traffic information to select a preset signal plan according to the best match with the detected traffic pattern pertain to *plan selection*. Several well-known signal calculation packages are reviewed here; their characteristics are summarised in Table 1.

SCOOT (Hunt et al., 1982) and SCATS (Luk, 1984) are basically online variants of off-line optimisation strategies. Manual engineering work is required to update traffic data and feed them into an off-line optimiser, for example TRANSYT (Vincent et al., 1980), for the preparation of a library of plans that apply to different periods of a day and days of a week. The ultimate performance of such systems depends on the accuracy of the database and its conformity to software requirements. The online capability then enables the selection of the most appropriate plan from the library according to detected traffic, adjusts offsets between adjacent intersections to facilitate traffic flow, and makes small adjustments to the signal plan. SCOOT

Table 1
Summary of different design programs for traffic signal control.

Program	Traffic data	Decision on signal settings	Signal cycle	Signal coordination	Origin country	Objective for optimisation	Servo mechanism
OPAC	Online data from upstream detectors	Change of current signal settings Rolling forward	Acyclic	Through traffic profile	USA	Delay	Decentralized
UTOPIA	Online data from upstream detectors	Green start times, durations and offsets	Required	With offset optimisation	Italy	Stops and delay	Centralized
SCATS	Online data from stop line (downstream) detector	Pre-calculated signal plan selection	Required	With offset optimisation	Australia	Capacity*	Centralized
SCOOT	Online data from upstream detectors	Adjustment of whole signal plan	Required	With offset optimisation	UK	Stops, delay and congestion	Centralized
PRODYN	Online data from pair of upstream detectors	Change of current signal settings	Acyclic	Possible	France	Total delay	Decentralized
MOVA	Online data from a single upstream detector	Green extension or not	N/A	Nil	UK	Stops, delay and capacity	Decentralized
DYPIC	Off-line basis and perfect information	Complete signal settings	Acyclic	Nil	UK	Delay	Decentralized

* Vehicle discharge rates are monitored, and signal will be changed if it is found less than the saturation flow rate.

reduces delay to vehicles by 12% when compared to the plans from TRANSYT. SCATS produces 23% reduction in travel time in comparison with uncoordinated operations.

DYPIC (Robertson and Bertherton, 1974), OPAC (Gartner, 1983) and PRODYN (Henry et al., 1983) are successive developments in dynamic traffic signal controller. DYPIC is actually a backward dynamic programming approach that serves only for analytical purpose. An empirical function of quadratic form is derived from the DYPIC study to form a key feature of a heuristic solution intended for practical uses. The heuristic solution adopts the concept of rolling horizon, which implies that: first, a planning horizon is split into a 'head' period with detected traffic information and a 'tail' period with predicted traffic information; secondly, an optimal policy is calculated for the entire horizon, but is only implemented for the 'head' period; finally, when the next time step arrives and new information becomes available, the process rolls forward and repeats itself. Gartner (1983) provides a detailed description of rolling horizon approach in his study of OPAC. However, OPAC does not abide by the principle of optimality adherent to dynamic programming; rather it uses optimal sequential constrained search (OSCO) to plan for the entire horizon, and employs terminal cost to penalise queues remaining in the system after the horizon. OPAC in both simulation and field tests reduces 5–15% vehicle delay from existing traffic-actuated methods, with most of the benefits coming from situation of high degree of saturation. The concerns with OPAC are that the restrictions in OSCO search reduce the flexibility of decision making, and a long planning horizon (60 s) raises practical questions about optimisation far into the future on the basis of predicted information, when the decisions planned for the 'tail' may never the implemented. PRODYN, also adopting rolling horizon approach, optimises timings via a forward dynamic programming (FDP). To avoid computing Bellman's equation at many grid points that eventually poses the problem of dimensionality, the FDP is particularly designed so that it aggregates state variables into a few subsets, and the value of being in a subset is only evaluated when it is actually being arrived at. A value function presenting the future cost in the FDP is directly adopted from Robertson and Bretherton's work. By evaluating all the subsets that can be reached, the FDP calculates the optimal trajectory of control policy in the planning horizon (75 s). The process then rolls forward one step in time. Experiments (Henry, 1989) show that PRODYN yields an average gain in total travel time of 10% with at 99.99% significance.

UPTOPIA (Urban Traffic Optimisation by Integrated Automation) (Mauro and Di Taranto, 1989) is a hybrid control system that combines online dynamic optimisation and off-line optimisation. This is achieved by constructing a system hierarchy with an area level and a local level. The area controller generates reference plan, and local controllers adapt this reference plan and dynamically coordinate signals in adjacent intersections. The rolling horizon approach is again used by local controllers to optimise performance, and the planning horizon is 120 s, with the process being repeated every 3 s. To automate the process of updating reference plans, which are generated by TRANSYT, an AUT (Automatic Updating of TRANSYT) module is developed. AUT first collects traffic data continually from the detectors in the network. The data are processed to calculate traffic flow pictures for different periods of the day. The model predicts the traffic flow profiles for calculating new reference plans. Afterwards, AUT prepares the data for TRANSYT calculation and starts TRANSYT optimisation for selected effects. The benefits recorded after the implementation of UTOPIA show an increase of 15% in average speed for private vehicles and 28% for public transport with priority.

MOVA (Vincent and Peirce, 1988) is the only one in this review package which is purposely designed for isolated intersection. The system generates signal timings cycle-by-cycle. The timings vary continuously according to the latest traffic condition. Upon changing signal stage, MOVA uses vehicle gap detected through pairs of upstream detectors to terminate green extension. The criterion for extension is whether the gap reaches certain critical values. There are two operational modes specified for uncongested and congested conditions. In the uncongested mode, delay and stop are minimised, while in the congested mode, capacity is maximised. MOVA evaluates its signal plans every half second.

From the reviews on the existing online traffic signal control systems, a scope for new development in this field is perceived as the follows.

First, the controller at a single intersection adjusts signal timings dynamically with a larger freedom so long as it conforms to the prevailing traffic condition and facilitates the traffic flow in the network. Certain critical constraints such as minimum and maximum green time are respected. This would allow the controller to give green indication regardless of any preset stage sequence, duration and cycle time, and hence greater flexibility in responding to variations in traffic flow. The local controllers of OPAC and PRODYN are already capable of doing this, but a general formulation is yet to be shown for a multiple signal-stage intersection, and the fundamentals of DP should be reaffirmed. The key issue here is how to build an algorithm based upon Bellman's equation in a forward process (thus using limited online information), while avoiding computational difficulties. As optimality in control is not particularly sensitive to variations in traffic arrivals beyond 25 s in the future (Robertson and Bertherton, 1974), the rolling horizon does not need to exceed this.

Secondly, controller may automatically adjust control parameters, so that it adapts to changes in the prevailing traffic conditions. Although UTOPIA is capable of automating the update of the TRANSYT plans at area level, its local controllers are still subject to system-wide reference plans.

Thirdly, rapid revision of signal plans confers advantages over slower revision. This is because what could be addressed with a slower rate of revision will remain possible with a faster revision rate, but not vice versa. MOVA's resolution level has reached 0.5 s, but its timing plan is still cyclic.

The objective for this study therefore is to develop a practical approach based on ADP that adopts the fundamentals of the DP and its control parameters and revises its plans at a fast rate, preferably every 0.5 s. Application of the new approach is limited to isolated intersection with multiple stages in this study. However, this does not necessarily limit its potential in distributed control of traffic networks.

3. Approximate dynamic programming

In principle, ADP systems should be able to approximate the solution to any problem in control or planning which can be formulated as an optimisation problem (Werbos and Pang, 1996). In this study, we seek an approximation of the true value function in dynamic programming for controlling traffic signals. Discussion on ADP formulation starts from the fundamentals of the DP in Section 3.2. We show in Section 3.2 that the DP's usefulness is constrained by dimensionality of state space and availability of information, and in Section 3.3 that the ADP approach reduces the computational burden while being effective in using limited online information. The open framework of ADP allows various learning techniques to be employed for updating approximate function online. Under the concept of reinforcement learning, two specific learning techniques, i.e. temporal-difference learning and perturbation learning, are discussed, respectively in Section 3.4 and Section 3.5. System dynamics are introduced in Section 3.6.

3.1. Notation

i is a vector of system state,
$J(i)$ is the true value function associated with state i,
$\tilde{J}(\cdot, r)$ is an approximate function of $J(i)$,
r is a vector of functional parameters,
Δr is a new estimate of r,
$f(\cdot)$ is a function that returns Δr,
u is a decision vector,
w is a column vector of traffic arrival information,
α is a discount factor
$e^{-\theta t}$ is the discount function,
θ is a discount rate for cost incurred in the future,
$g(\cdot)$ is a one-step cost function,
l is a vector of traffic state,
s is a vector of controller state,
$y(w, i, u)$ is a function that returns a vector of vehicle departures,
$\phi(\cdot)$ is a feature-extraction function,
Φ is a vector of $\phi(\cdot)$.

3.2. Dynamic programming

Let $i \in X$ be the state variable of the system, and $u \in U$ the decision variable. Given the initial state i_0 and a sequence of decisions u_t at discrete time t, a dynamic programming algorithm is to solve

$$\min_{u_t \in U} E_{w_t}\left\{\sum_{t=0}^{T-1} \alpha^t g(i_t, i_{t+1}) | i_0 = i\right\}, \tag{1}$$

by recursively calculating

$$J(i_t) = \min_{u_t = U} E_{w_t}\{g(i_t, i_{t+1}) + \alpha J(i_{i+1}|i_t)\}, \text{ for } t = T-1, T-2, \ldots, 0. \tag{2}$$

where α is the discount factor that is defined by

$$a^t = \exp(-\theta t), \tag{3}$$

and T is the duration of planning period. Eq. (2) is also often noted as the *Bellman's equation*.

In the case of traffic signal control, a step t is a discrete time interval along the planning period, and a state i_t is a combination of traffic state l and controller state s. State l of an isolated intersection can be defined by the number of vehicles queuing in each of the approaching links; state s can be defined by the signals conditions including green and red indication, any changes currently underway, the times at which changes will be completed and the times of expiry of any minimum or maximum permitted durations. Action space associated with a state may include whether to change the current indication, and if so, which signal stage is to be called upon.

Although (2) offers a simple and elegant way of problem solving, it can be computationally intractable even for very small problems. The reason is that the algorithm has to loop over the entire state space to evaluate the optimal decision at a single step. The computational load therefore increases exponentially with additional state spaces. For example, in developing PRODYN, Henry et al. (1983) have found that the memory requirements for an intersection are

$$2\left[\Delta l\left(\prod_{n=1}^{N_l} C_m^1\right) T_g N_s\right](\omega_1 + \omega_2),$$

where 2 stands for the current and future tables of state values, Δl is the unit vehicle length, N_l is number of links, C_m^1 is a combination with a subset size of 1, and a set size of m, T_g is green time already elapsed, N_s is number of signal stages and ω_1 and ω_2 is the memory requirement for traffic state and controller state, respectively.

If an intersection has four links in two signal stages ($N_l = 4$ and $N_s = 2$), and no more than 19 vehicles could possibly queue in a link, with $\Delta l = 1$ and a maximum green time of 60 s ($T_g = 13$ at a resolution of 5 s per time increment), $\omega_1 = 2$ bytes, $\omega_2 = 2$ bytes, the memory requirement is:

$$2(20^4 \times 13 \times 2)(2+2) \approx 33 \text{ MB}.$$

If the complexity of the intersection rose further, the computational demand may even prove too costly for up-to-date computing facility with microprocessor. Another concern that restricts DP's application in real-time control is that information from detectors is normally of the next few seconds. In a rolling horizon approach, traffic models may supplement information for the rest of the planning horizon, which is usually more than 60 s in previous studies. However, using DP means starting calculations at the end of the horizon where information on arrivals is least certain. This scenario puts difficulty in justifying the use of a costly optimisation algorithm in a process involving a large number of modelled information. Decisions planned in the 'tail' period may never be implemented.

3.3. Formulations of approximation of dynamic programming

Since the predominant source for the computational burden of the DP comes from calculating (2) for every single state, i, associated with a step t, we may seek to reduce the state space by introducing an approximate value function $\tilde{J}(\cdot)$ with continuous state variables. The computational requirement may then become polynomial to the number of state variables, rather than being exponential to the size of state space. By using an approximate value function, the value iteration that solves the DP problem takes a process that steps forward into time. In a forward process, the more reliable part of information is used to evaluate decisions accommodating the imminent traffic demand. Using reinforcement learning, the approximation to the true value function is updated upon each observation of state transition. The approximation function $\tilde{J}(\cdot)$, the forward process and the adaptation of approximation are the three distinctive features of the ADP approach.

To form mathematical formulation for the ADP approach, we use a continuous approximation function $\tilde{J}(\cdot, r) : X \times \mathbb{R}^{\mathbb{K}} \to \mathbb{R}$ to replace the true value function $J(\cdot) : X \to \mathbb{R}$, where parameter vector r of $\tilde{J}$ is K-dimensional. At each time step t, we calculate

$$\hat{J}(i_t) = \min_{u_t \in U} E_{w_t}\{g(i_t, i_{t+1}) + \alpha \tilde{J}(i_{t+1}, r_t)\}, \quad \text{for } t = 0, 1, \ldots, T-1 \tag{4}$$

and implement at each time step t

$$u_t^* = \arg\min_{u_t \in U} E_{w_t}\{g(i_t, i_{t+1}) + \alpha \tilde{J}(i_{t+1}, r_t)\}. \tag{5}$$

We calculate (4) only upon visiting the actual state i_t, thus substantially reducing computational requirement. Without knowing the approximate function *a priori*, we may hypothesize any structure that we deem as appropriate and employ a machine-learning technique to tune the functional parameters progressively. There are a large number of approximation techniques available, including aggregation, regression and most frequently neural networks. In this study, we limit the investigation on continuous linear approximation function only, which can be expressed as:

$$\tilde{J}(i, r) = \sum_{n=1}^{N} r'(n)\phi_n(i), \tag{6}$$

Using (6) to replace the true value function $J(i)$, the memory requirement for a four-link two-stages intersection is

$$2(4 \times 13 \times 2)(2+2) = 832 \text{ bytes}.$$

This amount is one 2.5×10^5th of what would have been in the DP. Furthermore, because of the forward process, the ADP algorithm starts decision making with the actual data detected from online sensors, and implements decisions only for the 'head' period in a rolling horizon approach. The head period is usually less than or equal to the period of actual data. The concern of starting planning backward from where data are least certain is then circumvented.

We may further assume that a certain learning paradigm supervises the update of parameter vector r whenever new observation $\hat{J}(i)$ becomes available. This can be expressed as:

$$\Delta r_{t+1} = f(\hat{J}(i_t), r_t). \tag{7}$$

Depending upon the specific learning technique in use, the objective of parameter tuning could be different, although a least square approach for minimising the error between estimation and observation is common. We summarise the general ADP algorithm in Fig. 1.

3.4. Reinforcement learning

Reinforcement learning is a paradigm of learning without a 'teacher'. It is a 'behavioural' learning problem: It is performed through *interaction* between the learning system and its environment, in which the system seeks to achieve a specific goal

Step 0. Initialization:

a. Initialize r_0,

b. Choose an initial state i_0,

c. Set t=0.

Step 1. Receive random information w_t.

Step 2. While $t <= T$,

a. Solve Eq. (5) to find u_t^*.

b. Update approximation through Eq. (7).

c. Implement u_t^*, and transfer the current state i_t to the new state i_{t+1}.

Step 3. Let $t = t + 1$. If $t < T$, go to step 1, otherwise stop.

Fig. 1. A general approximate dynamic programming algorithm 47.

despite the presence of uncertainties (Barto et al., 1983; Sutton and Barto, 1998). Trial and error searches are distinguishing features of reinforcement learning.

A reinforcement learning agent generally consists of four basic components: *a policy, a reward function, a value function*, and *a model of the environment*. In a problem defined by (1) and (2), the policy is the Bellman's equation shown in (2). The policy is the ultimate determinant of behaviours and performance. The reward function is $g(\cdot)$, which returns the immediate and defining feature of the problem faced by the agent. The value function is represented by $J(i)$, which estimates the rewards in the long run. The model of the environment can be the system that transfers state i_t to i_{t+1}. It is not difficult to find that the DP formulas are the basis for a reinforcement learning problem.

In an approximation that seeks to replace the exact $J(i)$ with a function approximation shown in (6), reinforcement learning technique updates the functional parameters upon each observation of a state transition. The objective is to improve approximations as more state transitions are observed. The relationship between the components of a reinforcement learning agent is shown in Fig. 2. The specific learning techniques employed in this study include *temporal-difference learning* (TD learning) and perturbation learning.

The TD learning, originally proposed by Sutton (1988), is a method for approximating long-term future cost as function of current state. Let us consider an irreducible aperiodic Markov process of infinite horizon whose states lie in a finite or countable infinite space X. The true value function $J : X \to \mathbb{R}$ associated with this Markov process is defined by

$$J(i) \stackrel{\Delta}{=} \mathrm{E}_w\left\{\sum_{t=0}^{\infty} \alpha^t g(i_t, i_{t+1}) | i_0 = i\right\}, \tag{8}$$

assuming that the expectation is well defined. We consider a linear approximation of J using a function defined by (6). The temporal difference d_t is then defined as

$$d_t = g(i_t, i_{t+1}) + \alpha \widetilde{J}(i_{t+1}, r_t) - \widetilde{J}(i_t, r_t). \tag{9}$$

For $t = 0, 1, \ldots$, the TD learning method updates r_t according to the formula

$$r_{t+1} = r_t + \eta_t d_t \sum_{k=0}^{t} (\alpha\lambda)^{t-k} \nabla_r \widetilde{J}(i_k, r_t), \tag{10}$$

or taking account of our case of linear approximation

$$r_{t+1} = r_t + \eta_t d_t \sum_{k=0}^{t} (\alpha\lambda)^{t-k} \phi(i_k). \tag{11}$$

where η_t is a sequence of scalar step sizes that satisfy the following terms for convergence

$$\sum_{t=0}^{\infty} \eta_t = \infty, \tag{12}$$

and

$$\sum_{t=0}^{\infty} \eta_t^2 < \infty, \tag{13}$$

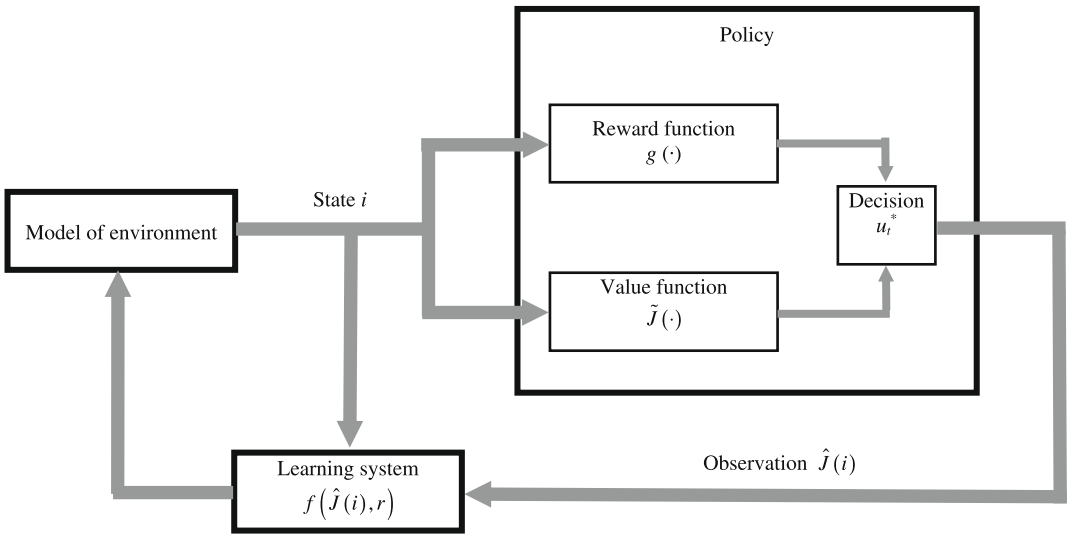


Fig. 2. A graphical illustration of the main components of a reinforcement learning agent, including a policy, a reward function, a value function, and a model of the environment; the arrows show the path of information in the system 48.

Parameter λ of (9) and (10) is known as *trace eligibility* factor, which takes value in [0, 1]. Since the TD learning is actually a continuum of algorithm parameterized by λ, it is often referred as TD(λ). A more convenient representation of TD(λ) can be obtained by defining a sequence of *eligibility vectors* z_t by

$$z_t = \sum_{k=0}^{t} (\alpha\lambda)^{t-k} \phi(i_t). \tag{14}$$

With this notation, Eq. (10) can be rewritten as

$$r_{t+1} = r_t + \eta_t d_t z_t. \tag{15}$$

For a better understanding of the nature of *trace eligibility* factor λ and the underlying dynamics of the TD algorithm, we may define a TD(λ) operator for $\lambda \in (0, 1)$ by

$$(T^{(\lambda)}J)(i) = (1-\lambda)\sum_{m=0}^{\infty} \lambda^m E\left[\sum_{t=0}^{m} \alpha^t g(i_t, i_{t+1}) + \alpha^{m+1} J(i_{m+1}) | i_0 = i\right], \tag{16}$$

and for $\lambda = 1$ by

$$(T^{(1)}J)(i) = E\left[\sum_{t=0}^{\infty} \alpha^t g(i_t, i_{t+1}) | i_0 = i\right] = J(i), \tag{17}$$

and for $\lambda = 0$ by

$$(T^{(0)}J)(i) = E[g(i_t, i_{t+1}) + \alpha J(i_{t+1}|i)]. \tag{18}$$

It can be seen that TD(1) is a true and an unbiased estimation of $J(i)$, whereas TD(0) is an equivalent to *single-pass algorithm* as all the calculations including the update of approximation are finished at the end of each forward pass. In the case of TD(0), only the most recent observation matters, which gives it a simplicity in a family of TD algorithms. Although TD(1) gives a quicker convergence than TD(0) in general, in the cases where states are frequently visited, TD(0) will work fine. Van Roy et al. (1997) developed an online ADP approach for inventory holding problem using TD(0). The approach reduces inventory costs by about 10% in comparison with optimised "order-up-to" policies.

With operator $T^{(\lambda)}J$, we can rewrite (10) in the form

$$r_{t+1} = r_t + \eta_t D(T^{(\lambda)}(r_t'\Phi) - r_t'\Phi)\phi(i_t), \tag{19}$$

where D is the diagonal matrix with diagonal entries $\pi(i)$, $i = 1, \ldots, n$,

$$D = \begin{pmatrix} \pi(1) & 0 & \ldots & 0 \\ 0 & \pi(2) & \ldots & 0 \\ & & \ldots & \\ 0 & 0 & \ldots & \pi(n) \end{pmatrix}.$$

The entries of D denotes the steady-state probability for the process i_t, and $\pi(i) > 0$, $\forall i \in X$. In the case of $\lambda \in [0, 1)$, we can regard (15) as a steepest descent method for a weighted linear least square problem as

$$\sum_{i \in X} \pi(i) \left(\left(T^{(\lambda)}(r'\Phi) \right)(i) - \tilde{J}(i, r) \right)^2 \tag{20}$$

with respect to r, given a fixed r_t. Tsitsiklis and Van Roy (1997) prove that a linear approximation function trained by TD(λ) converges with a probability of 1, if a few outstanding assumptions are met.

3.5. Perturbation learning

Regression models such as TD(λ) offers one way of approximating the value function of the DP. An alternative to regression model is to explore the structural property of the value function. A property of particular interest to the study is the monotonicity of the value function. In traffic signal control, we have a monotonic increase problem, i.e. the value of the scalar output of the value function increases as the number of vehicles remaining in the system increases. Additionally, the value increases more rapidly when vehicles are remained in red links than in green. In real-time operation, we may not have the complete knowledge of the true value function, but we can approximate the property of monotonicity. Since a linear approximation function with non-negative parameters is monotonically increasing, we may define a learning function on the basis of (6). The partial differentiation of the linear function with respect to $\phi_n(i)$ is

$$\frac{\partial \tilde{J}(i, r)}{\partial \phi_k(i)} = r(k),$$

and we can easily show that a new observation of $r(k)$ can be obtained through

$$\Delta r(k) = \frac{\hat{J}(i + \Delta i(k)) - \hat{J}(i)}{2\|\Delta i(k)\|}, \tag{21}$$

where $\Delta i\,(k)$ is a K-dimension vector of zeroes with a unit increment in the kth element, and $\hat{J}(\cdot)$ is calculated by (4). With the new observation, we then smooth to obtain an updated estimate of the functional parameter

$$r_{t+1}(k) = (1 - \eta_t) r_t(k) + \eta_t \Delta r_t(k), \quad \text{for } k = 1, 2, \ldots, K. \tag{22}$$

Using perturbation learning, Papadaki and Powell (2003) develop an ADP approach for stochastic batch dispatch problem, in which results from both the linear and non-linear (concave) approximations are compared. Although non-linear approximation works better than linear ones as iteration goes on, a large-scale problem may make non-linear approximation extremely difficult or impossible in implementation.

3.6. System dynamics

In this section, we show the formulations of system dynamics in the ADP algorithm. A state i of traffic control system is a combination of traffic state l and controller state s. We further define that traffic state l by the number of vehicles queuing in each of the approaching links, and controller state s by the state of signal (i.e. red or green, amber state is not considered in this study) of each link. For an intersection having total N links, for $n = 1, \ldots, N$, we define

$$l = \begin{bmatrix} l(1) \\ \vdots \\ l(N) \end{bmatrix}, \qquad s = \begin{bmatrix} s(1) \\ \vdots \\ s(N) \end{bmatrix},$$

where $l(N)$ denotes the actual number of vehicles queuing in link n, and each element of s is a binary variable depending on traffic signal indication such that

$$s(n) = \begin{cases} 1 & \text{if signal is green for link } n \\ 0 & \text{if signal is red for link } n \end{cases}$$

The system state i therefore can be expressed as i {l, s}. To construct the approximation function, we employ the feature-extraction function $\phi(i)$ such that,

$$\phi(i) = \begin{bmatrix} \phi(i(1)) \\ \vdots \\ \phi(i(N)) \end{bmatrix}, \quad \text{where } \phi(i(n)) = \begin{cases} \begin{bmatrix} l(n) \\ 0 \end{bmatrix} & \text{if } s(n) = 1 \\ \begin{bmatrix} 0 \\ l(n) \end{bmatrix} & \text{if } s(n) = 0. \end{cases}$$

The linear approximation function is formed by

$$\tilde{J}(i, r) = \sum_{n=1}^{N} r'(n) \phi_n(i),$$

where

$$r(n) = \begin{bmatrix} r^-(n) \\ r^+(n) \end{bmatrix}. \tag{23}$$

In such a way, we differentiate the signal status, and assign r^- to queue length variable $l(n)$ if link n receives green signal, or assign r^+ otherwise.

We further denote random arriving traffic by column vector w, where

$$w = \begin{bmatrix} w(1) \\ \vdots \\ w(N) \end{bmatrix}.$$

Let column vector y denote the departing traffic from the N-link intersection, so that

$$y = \begin{bmatrix} y(1) \\ \vdots \\ y(n) \end{bmatrix}.$$

Finally, the transition of system state during time increment from t to t+1 can be described as

$$l_{t+1}(n) = l_t(n) - y_t(n) + w_t(n), \tag{24}$$

and signal vector s is transferred

$$s_{t+1}(n) = (s_t(n) + u_t(n)) \mathrm{mod}_2, \tag{25}$$

where decision variable u_t takes

$$u_t(n) = \begin{cases} 1 & \text{for signal switch} \\ 0 & \text{unchanged} \end{cases} \tag{26}$$

Eqs. (24) and (25) describe the state transitions of a single step. The number of steps set in certain period depends on the resolution of the discrete time system. Let Δt denotes the time increment of discrete step, we assume that there is a total number of M steps in the planning period of the signal controller, and consequently the actual duration of the planning period is $M\Delta t$ s. Since we assume that the upstream detector provides 10-s data of future traffic, the planning period $M\Delta t$ is normally between 10 and 20 s. Traffic data for the period beyond 10 s can be predicted by using Monte Carlo simulation. For the M-step planning period, the ADP controller aims to find

$$\mathbf{u}_t^* = \arg\min_{u_t \in U} E_{w_k}\left[\sum_{k=t}^{t+M-1} \alpha^{k-t} g(i_k, i_{k+1}) + \alpha^M \widetilde{J}_{t-1}(i_{t+M-1}, r_{t-1})\right], i \in X, \tag{27}$$

and calculate

$$\hat{J}_M(i_t) = \min_{u_t \in U} E_{w_k}\left[\sum_{k=t}^{t+M-1} \alpha^{k-t} g(i_k, i_{k+1}) + \alpha^M \widetilde{J}_{t-1}(i_{t+M-1}, r_{t-1})\right], \quad i \in X, \tag{28}$$

where the one-step cost function g is given by

$$g(i_t, i_{t+1}) = \sum_{n=1}^{N} [l_t(n) - y_t(n) + w_t(n)]\Delta t. \tag{29}$$

The M-step temporal difference can be expressed as

$$d_M = \hat{J}_M(i_t) - \widetilde{J}(i_t, r_t) = \sum_{k=t}^{t+M-1} \alpha^{k-t} d_k(i_k, i_{k+1}), \tag{30}$$

and the parameters are updated by

$$r_{t+1} = r_t + \eta_t \phi(i_t) \sum_{k=t}^{t+M-1} \alpha^{k-t} d_k(i_k, i_{k+1}), t = 0, 1, \ldots. \tag{31}$$

Eq. (30) can be regarded as a special variant of (10). With a large M, Eq. (30) comes closer to (10) with $\lambda = 1$, and a smaller M makes (30) closer to (10) with $\lambda = 0$.

For the perturbation technique presented by (20) and (21), in the M-step planning, we simply use (27) instead of one-step Eq. (4) to calculate $\hat{J}_M$.

The traffic signal control algorithm using ADP can be summarised as the following:
Step 0: Initialisation

0.1 Choose an initial system state i_0;
0.2 Initialise functional parameter vector r_0;
0.3 Initiate learning rate (or step size) η_0;
0.4 Set time index $t = 0$.

Step 1: Receiving new information

1.1 Set time index $t = t + 1$;
1.2 Receive detected information w_t ;
1.3 Predict the information vector w'_t for the extra part of the planning period, if necessary.

Step 2: Evaluate control decisions

2.1 If signal change is not admissible, set $\mathbf{u}_t^* = 0$;
2.2 If signal change is admissible, for the planning period of M-steps, find the optimal decision $\mathbf{u}_t^*$ using (26).

Step 3: Update approximation function

a) The TD option:

3.a.1 Calculate new observation $\hat{J}_M(i_t)$ using (28)
3.a.2 Calculate current approximation $\widetilde{J}_{t-1}(i_t, r_{t-1})$ using (23);
3.a.3 Calculate M-step temporal difference using (30)
3.a.4 Update functional parameter vector r_{t-1} using (31).

b) The Perturbation option:

3.b.1 Numerically calculate partial gradient for $n = 1, 2, \ldots, N$ by perturbing queue $l_t(n)$ of state i_t by $\Delta l(n)$,
if $s(n) = 0$ (green signal in link n), using (21) and (28) to obtain $\Delta r_t^-(n)$,
if $s(n) = 1$ (red signal in link n), using (21) and (28) to obtain $\Delta r_t^+(n)$,
3.b.2 Update the parameter vector r_{t-1} accordingly by using (22) for $n = 1, 2, \ldots, N$,

Step 4: Implement optimal decision u_t^* for the first Δt of the planning period

4.1 Transfer signal status using (25);
4.2 Transfer queue status using (24);
4.3 Complete the state transition from i_t to i_{t+1}.

Step 5: Stopping Criteria

5.1 If $t < T$, then goes back to Step 1; Otherwise, stop.

In the next section, we discuss the application of the ADP control algorithm in numerical experiments.

4. Experiments and results

This section is organised as follows. We first introduce the method to generate random traffic data for numerical experiment in computer simulation in Section 4.1, followed by the design of testing intersection in Section 4.2, the control policy of the ADP controller in Section 4.3 and the traffic flow settings in Section 4.4. The general simulation environment is introduced in Section 4.5. Comparisons between DP, ADP and fixed-time performances are shown in Section 4.6. From these comparisons, we are able to investigate the advantages of the ADP controller in online operation and the influence of different learning techniques in performance.

4.1. Traffic data

Traffic data are assumed being detected from inductive loops imbedded upstream of the stop line. The detected information contains arriving traffic of the next 10 s for each traffic link. The detected information is assumed becoming available at the beginning of each time interval. In a coarser resolution of 5-s per time increment, traffic rate takes integer value of 0, 1, or

2 vehicles only. In a finer resolution of 0.5-s per time increment, traffic rate is a binary variable taking either 0 or 1. Other assumptions on traffic data are:

1. Approaching traffic is homogenous.
2. Queue lengths are calculated at the end of each time interval, neglecting the detail of vehicle behaviour during the interval.
3. Signals may only be switched at the boundary between intervals.

The random process for the coarser resolution is generated by binomial distribution. The maximum rate of two vehicles per time increment ensures that vehicle arriving rate could not exceed the capacity of the traffic link. Let Q be the random traffic rate, we denote the cumulative distribution function by

$$F(q) = P\{Q \leqslant q\}$$

and probability of $Q = m$, where m takes integer value of 0, 1, or 2, is

$$P\{Q = m\} = \binom{n}{m} p^m (1-p)^{n-m}, \quad m = 0, 1, \ldots, n. \tag{32}$$

Simulation of random arrival process is then realised by using *Inverse Transformation Method* (ITM), which first generates a *uniform random number* v between 0 and 1, and then set $F(q) = v$ and solve for q to produce the desired random observation from the probability distribution.

To extend the random process to the finer resolution, a shifted Bernoulli process is adopted. With probability P, an arrival is generated, and the trial has duration $T_a = n_b \Delta t$, with probability $(1 - P)$; if no arrival is generated, the trial has duration $T_a = \Delta t$. The mean number of vehicles generated in a single trial is $E(n) = P$, and the mean duration of the trial is

$$E(T_a) = [Pn_b + (1 - P)]\Delta t. \tag{33}$$

An estimate of the mean rate of traffic generation is given:

$$q = \frac{E(n)}{E(T_a)} = \frac{P}{[1 + P(n_b - 1)]\Delta t}, \tag{34}$$

The probability required to generate a certain mean arrival rate can be deduced by inversing the expression:

$$P = \frac{q\Delta t}{1 - (n_b - 1)q\Delta t}. \tag{35}$$

The rest of the process for generating random traffic at the finer resolution is identical to that of the coarser resolution. Specifications for generating traffic data in both cases are summarised in Table 2. It is worth noting that because queue lengths are calculated at the end of each interval, the vehicle delays measured at the coarser resolution are not directly comparable to delays measured in the finer resolution. In particular, in the case of the coarser resolution, no delay is attributed to either vehicle in a time increment during which two vehicle depart, while at the finer resolution, delay would occur to one of the vehicles for the whole of the departure time of the other. To facilitate the comparison between the two resolutions, we record the decision sequence of the coarser case and implement them in the finer case. The results are then comparable.

4.2. Traffic intersection

To investigate the capability of the ADP controller in operating multiple stage traffic intersection, we design an isolated intersection with three signal stages, as shown in Fig. 3. We also assume that:

1. Signal phases are composed of effective greens and effective reds only, and no amber interval is considered.
2. There are no constraints on the maximum duration of a green period. Both inter-green and minimum green are set at 5 s.

The saturation flow rate of all traffic links is two vehicles per 5 s. This is equivalent to 1440 vehicles per hour, a rate that is sufficiently close to the saturation flow rate of a single traffic lane.

Because we only consider an isolated intersection, there is no treatment of the physical length of traffic queue. The queue in a traffic link will not affect approaching traffic or the detectors that are assumed being located sufficiently upstream.

Table 2
Generating traffic data.

	Time interval	Traffic rate	Random process	Simulation
Coarser resolution	5 s	0, 1, or 2	Binomial	ITM
Finer resolution	0.5 s	0 or 1	Shifted Bernoulli	ITM

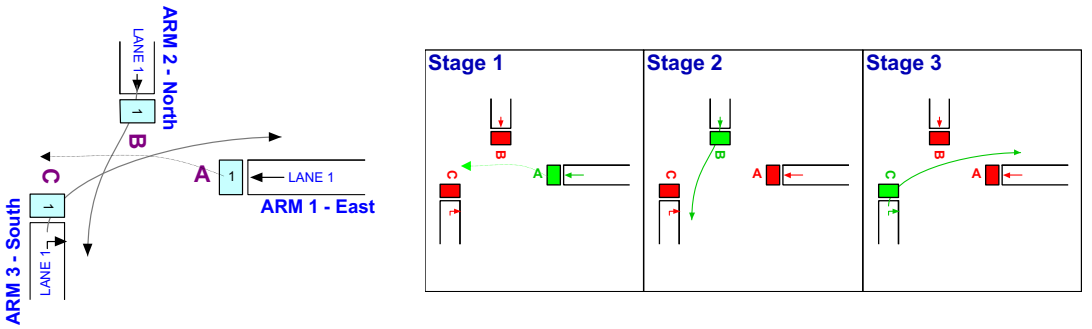


Fig. 3. The layout of the three-stage isolated intersection and a sample of fixed-stage sequence; each arm of the intersection has one traffic link, and the link has one traffic lane; the green in a link shows that the link receives green signal 49.

It is worth mentioning the treatment of *lost time*. Lost time in traffic engineering pertains to the time (in seconds) during which vehicles receiving green signal are unable to pass through an intersection. The total lost time is the sum of *Start-up lost time* and *Clearance lost time*. If no specific observations were made for the lost times, the two elements of total lost time may assumed to be 2 s (Roess et al., 2004). The lost time is not modelled in this study for reasons of simplicity.

4.3. Control policy

The ADP controller considers the following optional decisions:

(a) The signals are not changed
(b) The signals change immediately to the signal stage that gives least total delay
(c) The signals change in $k\Delta t$ s, for $k = t + 1, t + 2, \ldots, t + T_h - 1$, to the signal stage that gives least total delay, where T_h is the total number of time increments in the head part of the planning period.

The signals are changed only if decision (b) gives less total delay than both decisions (a) and (c). Using rolling horizon approach, only the first Δt is implemented.

In the case of the finer resolution, the evaluations of optional decision (c) require a 'tail' part of the planning period. This is because we want the planning period to be long enough to avoid all-red situation as the terminal signal state. For example, in evaluating decision (c), the decision of signal change at time $k = t + T_h - 1$ results in a mandatory inter-green of 5 s. This means that if the total number of steps in planning period M is equal to T_h, the terminal signal state at $k = t + T_h$ will be all-red, as the inter-green state demands. Given a planning period of M-step, terminating at all-red state is biased against decision (c), as all other options terminate in states with one stage in green.

In considering the length of the 'tail' part of the planning period, we prefer to keep it from being longer than 10 s. The reasons are that, firstly, Robertson and Bertherton (1974) use DYPIC to show that the optimal policy is not particularly sensitive to variations in the traffic arrivals of the next 10–20 s; secondly, a longer tail part swifts the weight from head to tail, thus from detected information to predicted information. By defining the tail part of being 10 s, we have a total planning period of 20 s, equivalent to 40 time increments in the case of finer resolution. The control policy is graphically illustrated in Fig. 4.

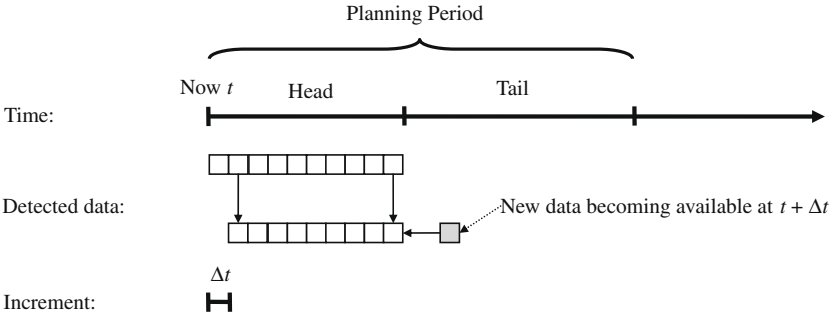


Fig. 4. The rolling horizon approach; the horizon contains a 'head' part with detected information and a 'tail' part with predicted information; the controller evaluate a decision for the entire horizon but implement the first Δt; the system then rolls forward and repeats the routine 50.

The 'tail' part is not considered at the coarser resolution, and the horizon for planning is 10 s, equivalent to two increments. The same control policy applies.

4.4. Traffic flows

We consider two scenarios of traffic in experiments. Scenario A has a constant hourly rate for each traffic link, and the specifications are 432 v/h for Link A, 252 v/h for Link B, and 432 v/h for Link C. In Scenario B, a profile of traffic is generated for each link. Each profile of traffic contains three periods: a pre-peak period, a peak period, and a post-peak period, as shown in Table 3 and Fig. 5, respectively.

4.5. Computer simulation

All the numerical experiments are conducted in computer simulation. The simulation program is built on MATLAB 7.4.0, and runs on a PC configured with Pentium® Dual Core 3.60 GHz and 3.50 GB of RAM. Function 'rand' of MATLAB is called to generate uniformly distributed random number, which is the core of traffic flow simulation. The TD learning capacity is provided by *neural network toolbox* (V5.0.2). We use a linear neuron to form the linear approximation function. Update of neuron weights (i.e. the parameter vector r) is performed by 'adapt' function. This approach of achieving TD learning in the ADP has been suggested by Betsekas and Tsitsiklis (1995), who call it *Neuro-Dynamic Programming*. A concern of using neural network to provide learning capacity is the stepsize η_t. A common approach in stochastic approximation is to use a time-varying rate:

$$\eta_t = \frac{a}{t}, \tag{36}$$

where a is a constant. It can be easily shown that this time-varying form satisfies the terms of convergence set by (11) and (12). However, the large learning rate at the beginning of the learning process may make the system prone to 'overshooting', which blows up the parameters. Moreover, in TD learning, all the elements of the parameter vector are updated at once, and there may not be a single rule that is right for all of the parameters. For these concerns, we use a constant and cautious

Table 3
Traffic Scenario B: a profile of traffic containing a pre-peak period, a peak period, and a post-peak period.

	Link A (v/h)	Link B (v/h)	Link C (v/h)
Pre-peak	250	150	250
Peak	500	250	500
Post-peak	350	200	350

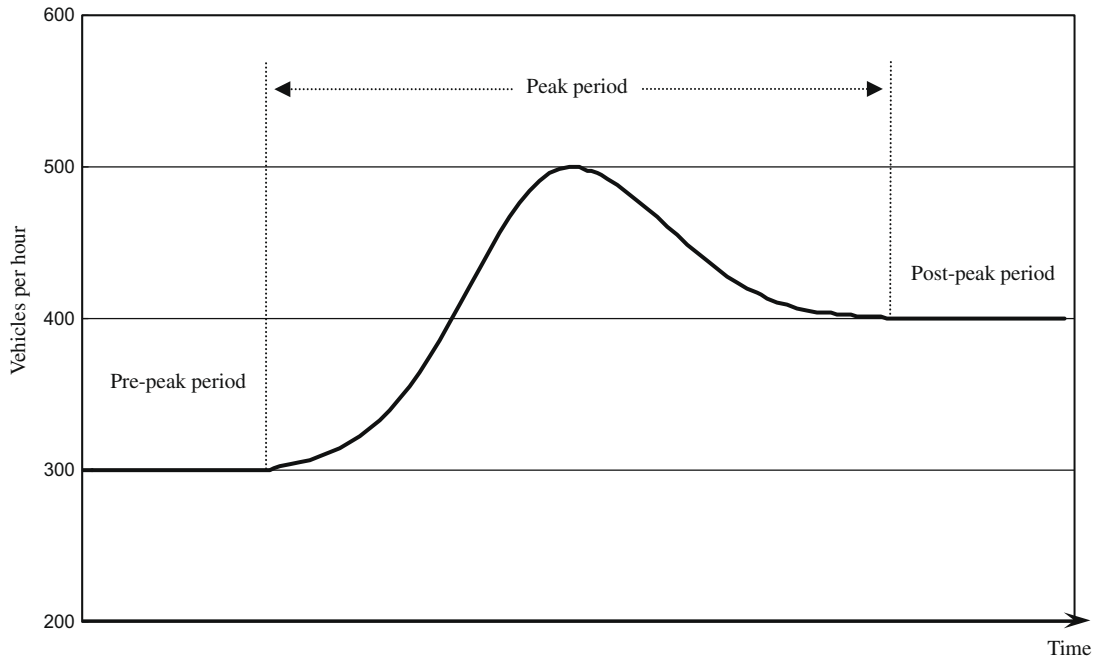


Fig. 5. Traffic Scenario B: a traffic flow profile containing pre-peak, peak and post-peak periods, with the peak occurs in the middle of the simulated time; the sample shows that for Links A and C the transfer between different traffic levels is smooth; the same smooth transfer also applies to Link B 51.

learning rate at $\eta = 0.001$ for updating functional parameter r. In the case of perturbation learning, because of the design of the algorithm, we are relieved from the concern of overshooting, and each parameter can be assigned a specific stepsize. For each parameter, we use

$$\eta_n = \frac{a}{a+n-1}, \tag{37}$$

where $a = 40$, and n counts the number of times the parameter has been updated (note that we use n rather than time index t here).

4.6. Results and comparison

With traffic Scenario A, in which the hourly traffic rate on each link is constant and the degree of saturation of the intersection is high, we compare the results from different control strategies, including DP, ADP and fixed-time. The DP approach provides the benchmark on the higher end of performance, whereas the fixed-time approach provides the benchmark on the lower end. In evaluating ADP performance, we obtain results from both the coarser resolution at 5 s per time increment and the finer resolution at 0.5 s. It is also of our interest to investigate whether ADP with different learning technique performs differently.

With traffic Scenario B, in which a profile of traffic is generated, we investigate the evolution of functional parameters in non-static environment and compare the influences of learning techniques. In the rest part of this section, we denote ADP with TD learning by ADP_TD, and ADP with perturbation learning by ADP_PL. All numerical experiments start from the same initial state that has no vehicles in system and green signal for Link A.

4.6.1. Traffic Scenario A

With Scenario A, we evaluate the ADP controllers' performance at both coarser resolution and finer resolution. In the case of the coarser resolution, only ADP_PL is evaluated against the best fixed-time plans. In the case of the finer resolution, both ADP_PL and ADP_TD are evaluated against benchmarks.

Since there is no specific method to determine the best value for parameter θ, which is associated with discount factor α by (3), we employ exhaustive search to find the optimum value. At the coarser resolution, we find that ADP_PL gives the best performance at $\theta = 0.04$. At the finer resolution, we find that at $\theta = 0.12$ both ADP_TD and ADP_PL report the best performance, as shown in Table 4. Using the same discount rate also ensures that comparisons between the two learning techniques are even-handed. Setting $\theta = 0.12$ means that the delays incurred in operation are discounted at 12% per time increment, and 24% per second. This is a substantial magnitude in discounting.

The computational difficulties of the DP approach prohibits us from searching discount rate exhaustively. Rather, we directly use $\theta = 0.12$ for the DP approach, thus being consistent in comparisons. It is worth noting that the computer in use can only run DP calculation with 6-min real data in a single run of simulation, otherwise the computer reports 'out of memory'. The result obtained is actually an average performance over 10 independent simulations, with each of 6 min. The best fixed-time timing plans are produced by TRANSYT 12.0 and summarised in Table 5. By transplanting the ADP_PL decision sequence in the coarser resolution to operation in the finer case, we establish performance comparisons shown in Table 6.

Table 4
Pilot tests on parameter θ for discount factor α and the corresponding performances (v.s/s) from the ADP_TD and the ADP_PL controller at resolution of 0.5 s per time increment.

Parameter θ	ADP_TD	ADP_PL
0.05	4.49	4.40
0.10	4.48	4.37
0.11	4.48	4.43
0.12	4.38	4.36
0.13	4.38	4.43
0.14	4.46	4.43
0.15	4.33	4.42
0.20	4.42	4.50

Table 5
The optimised fixed-time plans from TRANSYT 12.0 for the three-stage traffic intersection. The timings listed in the table are the corresponding starting time of the signal stage in a cycle, for example Stage 1 starts from the 55th second of a cycle of 120 s.

Number of stages	Starting time			Cycle time
	Stage 1	Stage 2	Stage 3	
3	55	101	9	120 s

Table 6
Performance comparisons with Traffic Scenario A, the decision sequences in coarse resolution are transferred to the fine resolution in order to ensure consistence in comparison.

	DP	ADP_TD	ADP_PL	ADP_PL	TRANSYT
Resolution	Fine	Fine	Fine	Coarse	Fine
Simulation time (min)	6	60	60	60	60
Run time (min)	720	5.5	12	0.3	1/6
Discount rate θ	0.12	0.12	0.12	0.04	–
Averaged performance (v.s/s)	4.27	4.62	4.66	8.64	13.95

At the resolution of 0.5 s, the ADP controllers reduce about 67% vehicle delays from the optimised fixed-time plans produced by TRANSYT. It is also worth noticing that the same ADP controller can reduce about 41% delays by operating at the finer resolution instead of at the coarser resolution. The higher frequency of revising signal plans proves rewarding.

The advantage of operating at the finer resolution comes from the ability to adjust signal timings more precisely according to the detected information of traffic. We depict a sample in Fig. 6, in which the signal sequences and queuing dynamics between time interval $t = 6300$ and $t = 6800$ are plotted and compared between the two resolutions. With identical traffic arrivals, the ADP_PL controller made 12 signal switches during the 250-s period at the finer resolution, and the signal sequence is:

B—C—A—B—A—C—A—C—B—A—C—B—A,

At the coarser resolution, however, the controller made nine switches, and the signal sequence is:

C—A—C—B—A—C—A—B—C—A.

The signal sequences are acyclic in both cases.

At the finer resolution, the ADP_TD controller performs as well as ADP_PL, but takes less than half the time in computing. The major contributor to the difference in computation comes from the algorithms for updating parameter r. The ADP_TD approach calculates a single observation $\hat{J}(\cdot)$ for all the parameters, whereas for every single parameter ADP_PL calculates a $\hat{J}(\cdot)$ with perturbation, as specified by (20). Nevertheless, the computational efficiency of both makes them fully capable for online operation. The ADP controllers in average only incur about 0.4 v.s/s more delay than the DP controller at the finer resolution. The DP controller takes an enormous amount of time (12 h or so) to finish a simulation of 6-min operation at an intersection of only three signal stages.

The large ratio in discounting future delay ($\theta = 0.12$) also means that the influence of the approximation function $\tilde{J}(i_{t+M}, r_t)$ in evaluating control decisions at the finer resolution is limited. We find that ignoring $\tilde{J}(i_{t+M}, r_t)$ in evaluation only worsens performance by 0.5 v.s/s, whereas taking a larger account of it has negative impact on performance. Therefore, it is of practical interest to remain the approximation function simple and easy to update.

4.6.2. Traffic Scenario B

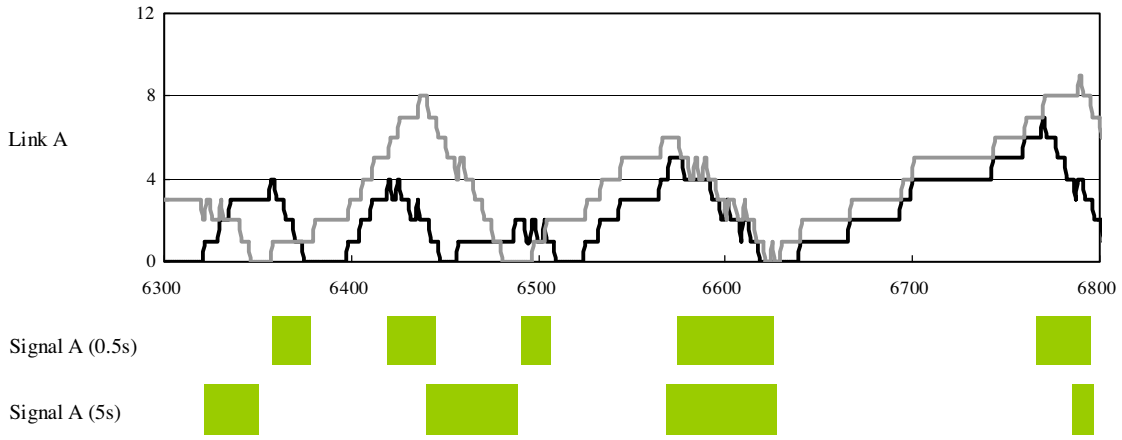
In this scenario of traffic input, we investigate the learning patterns of ADP in a changing and noisy environment and test whether different learning techniques result in distinguishable performance.

The evolutions of functional parameters updated by ADP_TD and ADP_PL in a single simulation run of 28,800 time increments at 0.5 s resolution, equivalent to 4 h simulated time, are shown in Figs. 7 and 8, respectively. Because of that the ADP_PL controller has time-varying learning rates that are large in the beginning, parameters start with strong oscillations. The magnitude of oscillation only begins to reduce when the system enters post-peak period, partly due to the diminishing learning rate and partly due to the steadier environment. The ADP_TD controller uses neural network that adopts a constant learning of small magnitude, i.e. at $\eta_t = 0.001$. Consequently, the parameters have less oscillation, and exhibit similar patterns over time to those updated by ADP_PL.

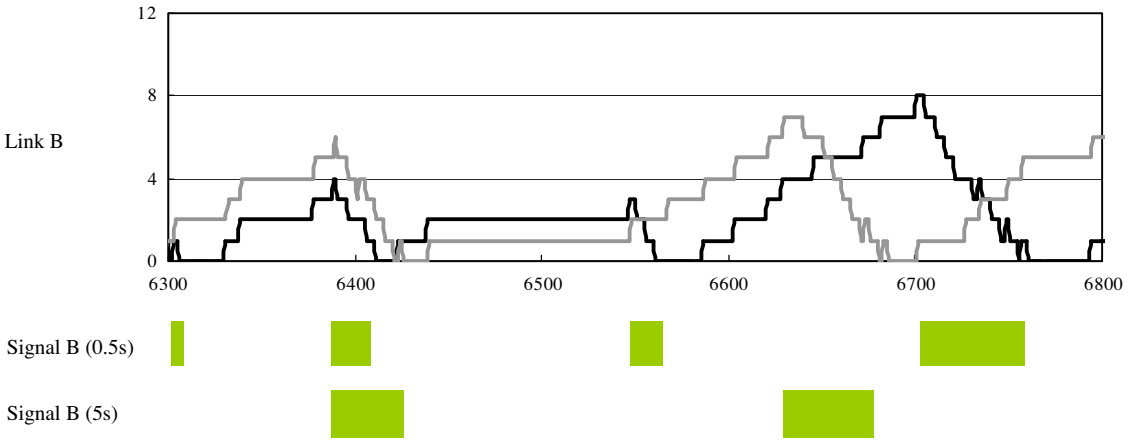
The rise of traffic volume in the peak period gives sharp rise in parameter values associated with green states, i.e. r^-, but parameter r^+ is largely insensitive to the changes in traffic across all traffic links, especially with ADP_PL. This pattern may partially explained by that parameter r^- of a traffic link is updated twice more frequently as r^+, as the link spent only one-third of the time in green state, and the rest of time in red state.

The comparisons in performance of the two learning techniques are recorded in Table 7. We use two-sample t-test to verify the hypothesis that the two means are equal. We find that the two means are equal at the degree of significance $\alpha = 0.05$, with a degree of freedom of $n = 9$. Recalling that ADP_TD is actually a regression method that minimises weighted square errors and ADP_PL estimates gradients numerically regardless of errors in prediction, the equality of the two means suggests that learning techniques have the same influence on the performance when the approximation function is linear, discount rate is large and decisions are frequently revised.

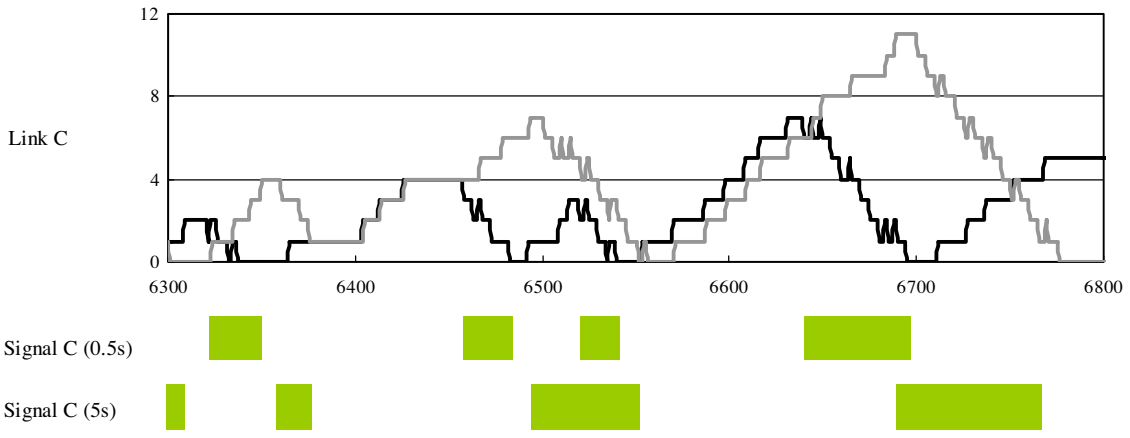
A general conclusion we can draw from the numerical examples is that the ADP controllers are fully capable of real-time operation at an isolated intersection. It reduces 67% of vehicle delays from the best fixed-time plan at a resolution of 0.5 s. It also shows that by operating at a resolution of 0.5 s, instead of 5.0 s, the same ADP controller can achieve 41% improvement in performance. For all of the benefits, the ADP controllers only use 10-s data of future arrivals while being efficient in



a The evolutions of queue (Y-axis) in Link A between time step 6300 and 6800 (X-axis)



b The evolutions of queue (Y-axis) in Link A between time step 6300 and 6800 (X-axis)



c The evolutions of queue (Y-axis) in Link C between time step 6300 and 6800 (X-axis);

Fig. 6. The samples of queue evolutions and signal sequences show the difference of operating the same ADP_PL controller at two resolution levels: the dark line plots queue at 0.5 s resolution and the grey line at 5-s resolution; the signal sequence shows the start and the end of green times 52.

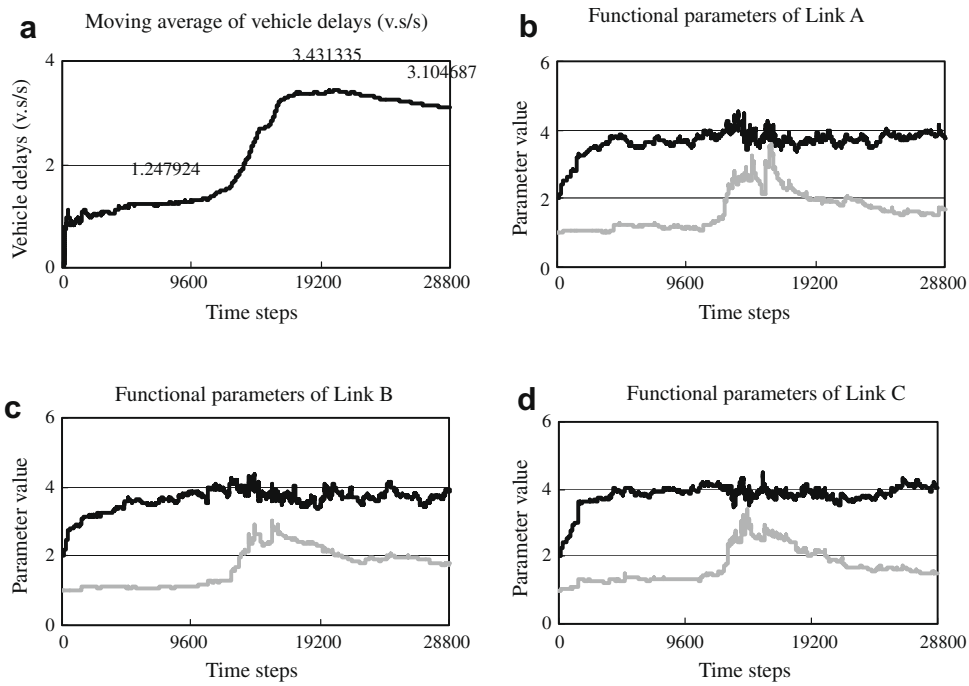


Fig. 7. The evolutions of functional parameters in traffic Scenario B, using the ADP_TD controller; the simulated time is 4 h, containing pre-peak, peak and post-peak periods; the parameters for green signal state respond to the rise in traffic with increase in value, while the parameters for red signal state are insensitive to traffic change 53.

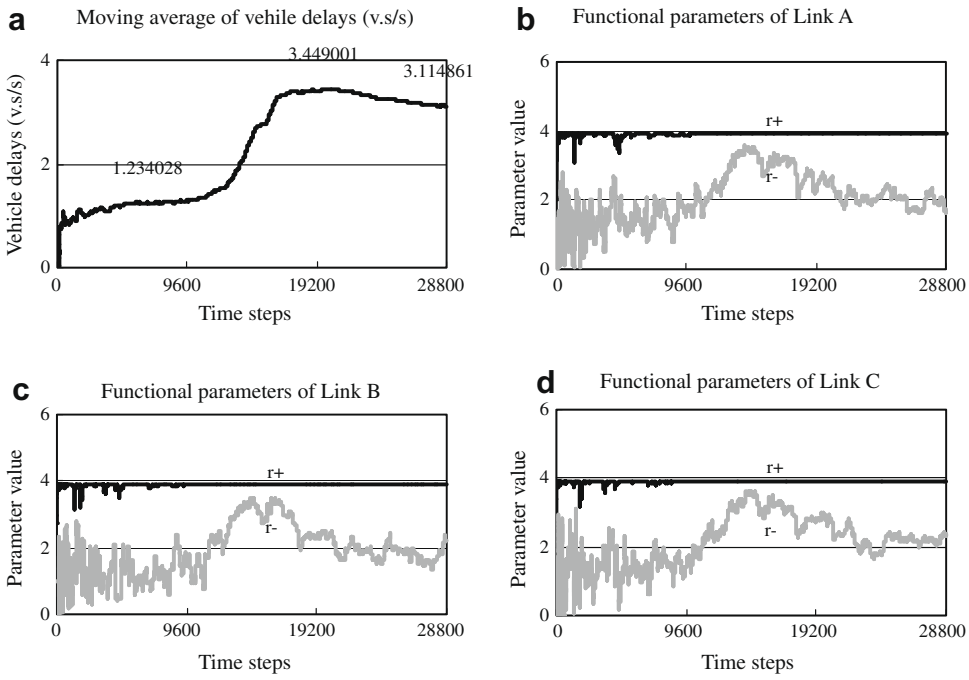


Fig. 8. The evolutions of functional parameters in traffic Scenario B, using the ADP_PL controller; the simulated time is 4 h, containing pre-peak, peak and post-peak periods; the parameters for green signal state respond to the rise in traffic with increase in value, while the parameters for red signal state are insensitive to traffic change; despite the difference in learning mechanism, the TD and PL learning have similar influence on the evolution of parameter and the performance 54.

computation. In a non-static traffic environment, the evolutions of the function parameters capture the changes in traffic environment. However, we find that the best performance of ADP is achieved at a relatively large discount rate. We also find

Table 7
Performance Comparisons with Traffic Scenario B, the performances of the two controllers in a single sample are recorded under identical traffic condition.

Sample	ADP_PL	ADP_TD
1	3.159583	3.376944
2	3.089722	2.974583
3	2.729583	2.623472
4	2.853194	2.985972
5	3.223472	3.3225
6	3.205278	3.184444
7	3.20375	3.286667
8	4.049861	4.266806
9	3.658611	3.522222
10	3.239028	3.215
Mean	3.24	3.28
t Stat		−0.819958009
P(T ⩽ t) two-tail		0.433417978
t Critical two-tail		2.262157158

that different learning techniques have the same influence in performance. It suggests that a simple linear approximation is sufficient for online operation, and the benefit of exploring complex approximation, such as non-linear functions, may not prove cost effective.

5. Conclusions

This study investigates the application of approximate dynamic programming (ADP) to the field of traffic signal control, aiming to develop a self-sufficient adaptive controller for online operation. Through the review of existing traffic signal control systems, we identify the objectives for this study as providing dynamic control for real-time operation, being adaptive to changing traffic by using online learning techniques, and frequent review of signal timing plans. We show in the numerical experiment that the ADP controllers meet all of the objectives. The key feature of the ADP approach is to replace the true value function in dynamic programming (DP) with a linear approximate function. The initial approximation is updated progressively by using specific learning techniques that adjust the function parameters in real-time. We have shown in the numerical examples that operating at a resolution of 5-s per time increment, the ADP controller reduces delay from 13.95 vehicle-second per second (v.s/s), which is the result from TRANSYT plans to 8.64 v.s/s. At the resolution of 0.5-s per time increment, the ADP controller reduces vehicle delay to 4.62 v.s/s. To provide an absolute lower bound in comparison, we provide results from DP after enduring a costly computation process, and the averaged result is 4.27 v.s/s under the same traffic condition that applies to the ADP controllers. The result from the ADP controllers at the finer resolution is just 0.4 v.s/s more than that of DP in average, whereas the time the ADP controller takes to complete an hour's simulation is only about 0.08% of the time the DP approach takes to complete 1/10 of an hour's simulation. The ADP controllers only use 10-s information of future arriving traffic in evaluating optional decisions. These results suggest that the ADP controller can achieve a large proportion of the benefits of DP, while being adaptive to the changing traffic and computationally efficient. The ADP approach is therefore a practical candidate for real-time signal control at isolated intersections.

Two learning techniques, i.e. temporal-difference (TD) reinforcement learning and perturbation learning, are investigated in this study. The TD method constantly tracks the different between current estimation and actual observation of state values, and propagates the difference back to the functional parameter so as to update the approximation. Perturbation learning directly estimates the gradients of the approximate function by giving a perturbing signal to the system state. Despite of the different learning methods, the learning effects are broadly similar, and no statistical difference can be found in numerical experiments. The ADP controller with either of the two learning techniques produces the best performance by discounting future delay at about 12% per time increment at the finer resolution, suggesting limited influence of the approximation function in evaluating control decisions. Combining the equality between two learning techniques with the limited influence, it suggests that a simple linear approximation is sufficient for the ADP controller to operate in real-time. Exploring more complex approximations may not prove cost effective.

The general formulations of the ADP controller presented in this study can be readily extended to more complicated intersections without substantial difficulty. However, the norm of contemporary traffic signal operation in urban areas is a coordinated network system, and achieving system-wide optimality is a common objective of these systems. This study has so far focused on isolated intersection only. Given the advantages of ADP controller shown in this study and its conformity to real-time operation, we identify the distributed network operation presented in OPAC and PRODYN as a good starting point for introducing ADP to traffic network. A challenging issue here is how to construct traffic state and controller state of adjacent intersections in the local objective function, and how the local controller could explore the power of ADP to learn from network operation.

Acknowledgements

The first author would like to thank Rees Jeffreys' Road Fund for the financial support for the work presented here. The second author would like to thank The Croucher Foundation for the support of the Croucher Fellowship. The work described in this paper was jointly supported by grants from the City University of Hong Kong (Project Number: 7001967 and 7200040) and the Research Grants Council of the Hong Kong Special Administrative Region, China (Project Number: 9041157). The authors thank the anonymous reviewers for their constructive comments.

References

Barto, A.G., Sutton, R.S., Anderson, C.W., 1983. Neuronlike elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics 13, 835–46. (Reprinted in Anderson, J.A., Rosenfeld E., 1988. Neurocomputing: Foundations of Research. MIT Press, Cambridge, MA.)

Bell, M.C., Bretherton, R.D., 1986. Ageing of fixed-time traffic signal plans. In: Proceedings of the Second IEE Conference on Road Traffic Control. IEE, London.

Bellman, R., 1957. Dynamic Programming. Princeton University Press, Princeton.

Betsekas, D.P., Tsitsiklis, J.N., 1995. Neuro-Dynamic Programming. Athenas Scientific, Belmont, MA.

Gartner, N.H., 1983. OPAC: a demand-responsive strategy for traffic signal control. Transportation Research Record 906, 75–81.

Henry, J.J., 1989. PRODYN tests and future experiment on ZELT, vehicle navigation and information systems conference. Conference Record, 292–295.

Henry, J.J., Farges, J.L., Tuffal, J., 1983. The PRODYN real time traffic algorithm. In: Proceedings of the fourth IFAC-IFIP-IFORS conference on Control in Transportation Systems, pp. 307–311.

Hunt, P.B., Robertson, D.I., Bretherton, R.D., 1982. The SCOOT on-line traffic signal optimisation technique. Traffic Engineering and Control 23, 190–192.

Luk, J.Y.K., 1984. Two traffic-responsive area traffic control methods: SCAT and SCOOT. Traffic Engineering and Control 25, 14–22.

Mauro, V., Di Taranto, C., 1989. UTOPIA. In: CCCT'89 – AFCET Proceedings, Paris.

Papadaki, K., Powell, W.B., 2003. An adaptive dynamic programming algorithm for a stochastic multiproduct batch dispatch problem. Naval Research Logistics 50 (7), 742–769.

Powell, W.B., 2007. Approximate Dynamic Programming: Solving the Curses of Dimensionality. John Wiley & Sons, Inc., Hoboken, New Jersey, ISBN 978-0-470-17155-4.

Robertson, D.I., Bertherton, R.D., 1974. Optimum control of an intersection for any known sequence of vehicular arrivals. In: Proceedings of the second IFAC-IFIP-IFORS Symposium on Traffic Control and Transportation system, Monte Carlo.

Roess, R.P., Prassas, E.S., McShane, W.R., 2004. Traffic Engineering, third ed. Pearson Prentice Hall, Upper Saddle River. ISBN 0-13-142471-8.

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

Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA.

Tsitsiklis, J.N., Van Roy, B., 1997. An analysis of temporal difference learning with function approximation. IEEE Transactions on Automatic Control 42 (5), 674–690.

Van Roy, B., Bertsekas, D.P., Lee, Y., Tsitsiklis, J.N., 1997. A neurodynamic programming approach to retailer inventory management. In: Proceedings of the 36th IEEE Conference on Decision and Control, IEEE, vol. 4. pp. 4052–4057.

Vincent, R.A., Peirce, J.R., 1988. MOVA: traffic responsive, self-optimising signal control for isolated intersections, Transport and Road Research Laboratory Report RR 170. TRRL, Crowthorne.

Vincent, R.A., Mitchell, A.I., Robertson, D.I., 1980. User guide to TRANSYT version 8. Transport and Road Research Laboratory Report LR888, Crowthorne, Berkshire, UK.

Werbos, P., Pang, X., 1996. Generalized maze navigation: SRN critics solve what feedforward or hebbian nets cannot. In: World Congress on Neural Networks, San Diego, CA. Lawrence Erlbaum INNS Press, pp. 88–93.