APPROXIMATE DYNAMIC PROGRAMMING CONTROLLER FOR MULTIPLE INTERSECTIONS

Cai, Chen

Le, Tung Mai

National ICT Australia

University of New South Wales

ABSTRACT

In this paper we propose a distributed control method based on approximate dynamic programming for traffic networks. A closed-form linear function is used to map state to scalar quantity as an approximation to the look-up table presentation of exact quantities used in dynamic programming. This substantially reduces computational requirement and makes the proposed method conforming to real-time operation. Temporal-difference learning is used to improve approximation at real-time. A cellular automation model is used to describe traffic dynamics in road networks. We show in numerical experiments that the proposed approach significantly improves control performance from optimised fixed-time plans in a range of test scenarios. Indicator for control performance is weighted sum of vehicle delays and stops.

Keywords: approximation, dynamic programming, distributed control

1. BACKGROUND

In typical urban traffic networks, control decision made at a single intersection affects traffic both upstream and downstream of the site. A common objective for controlling traffic networks is to ensure system-wide optimality in terms of performance measurements, including vehicle delays, stops, CO_2 emissions and travel reliability. Two approaches emerged to address this problem: one is centralised that views the system as a single entity; the other one is distributed and optimises local performance subject to upstream and downstream conditions. The former usually enforces coordination among adjacent intersections, and the latter relies on voluntary coordination. Examples of the former are TRANSYT (Vincent *et al.*, 1980), SCOOT (Hunt *et al.*, 1982), SCATS (Lowrie, 1992) and

UTOPIA (Mauro et al., 1989) systems, and those of the latter include OPAC (Gartner, 1982, 1983a, 1983b) and PRODYN (Henry *et al*., 1983). A hybrid system combining local dynamic optimisation and higher-level intervention on signal coordination is RHODES (Mirchandani and Head, 2001).

Due to the complexity associated with traffic network control, centralised systems usually do not include system-wide performance measures in the objective function. In stead, they identify key intersections of high degree of traffic saturation, and calculate optimised cycle time for the key intersections. The optimisation techniques often set to equalise degrees of saturation of all signal phases at the key intersection as an approximation to the optimal solution. Cycle time of the adjacent intersections are synchronised and pre-set offset plans are enforced to facilitate good propagation of flow from and to the key sites. SCOOT, SCATS and UTOPIA are examples of this. Disadvantages of this approach are that synchronised cycle time imposes constraints at local sites and pre-set offset usually does not take account of distribution and dynamics of traffic density in links.

The distributed control approach orients from the assumption that necessary upstream information for operating a local site is conveyed by the propagation of traffic flow. This is to say that if local controller was able to facilitate good propagation of flow from upstream and to downstream, then the local control policy is an approximation to optimal policy. Information of traffic flow can be detected using traffic sensors, and traffic model is required to establish sufficient presentation and projection of traffic state. The ideal optimisation technique for distributed control is dynamic programming (Bellman, 1957). This approach is the only exact solution to sequential decision-making of complex system so far. However, this technique faces difficulties for real-time application because of the computational requirement and demand of information (of traffic state, underlying model and vehicle arrivals). As the result, OPAC and PRODYN developed heuristics respectively to approximate the sequential decision-making process of dynamic programming (DP).

New development in approximating principle features of dynamic programming addresses the origin of computational difficulty. This usually involves approximation of the value function, state transition model or the policy of the DP so that computational demand can be reduced. These approaches are within the concept of approximate dynamic programming (ADP). Initial development of ADP for isolated intersection control was presented in Cai (2007) and Heydecker *et al*. (2007), and improved ADP with machine learning features was presented in Cai *et al*. (2009). These works were based function approximation. Policy approximation approaches to traffic signal control were seen in Teodorovic *et al*. (2006) and Li *et al*. (2008). A concept of using ADP controllers for distributed network control was proposed in Xiang *et al*. (2007), but was not seen in complete formulation and application. Their proposal is to use *artificial neural network* (ANN) to approximate the value function of DP.

In this paper, we present the formulae and numerical experiment results of applying ADP to distributed traffic network control. We propose a closed-form function that maps state to scalar quantity to approximate the exact value function of DP. The approximation function is

updated using real-time machine learning techniques. A cellular automaton model is used to describe vehicle behaviour at intermediate level, from which the system interpret traffic information and construct the traffic state. We apply the distributed ADP approach to a set a numerical experiments, and compare results with benchmarking control systems.

The rest of this paper is organised as the followings. In Section 2 we introduce the concept and formulae of ADP. The dynamics of traffic and control system at local intersection is introduced in Section 3. The distributed control structure and control policy for local operation are discussed in Section 4. Numerical experiments and results are presented in Section 5. A summary of this study is presented in Section 6.

2. APPROXIMATE DYNAMIC PROGRAMMING

2.1. Notations

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,
a	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,
W	is a weighting factor for vehicle stops
$y(.)$	is a function that returns a vector of vehicle departures,
z	is a vector of vehicles stops
$\phi(\cdot)$	is a feature-extraction function,
Φ	is a vector of $\phi(\cdot)$.

2.2. Dynamic programming and the approximation

Approximate dynamic programming is a derivative of dynamic programming (DP). It particularly addreses application of the principles of DP to make sequential decisions to solve complex problems. The principles of DP can be presented as the followings. Given the initial state i_0 and a sequence of decisions u_t at discrete time t, a DP algorithm is to solve

$$\min_{u_t \in U} E\left\{\sum_{t=0}^{m-1} \alpha^t g\left(i_t, i_{t+1}\right) \middle| i_0 = i\right\}. \quad (1)$$

The backward dynamic programming solution recursively computes the *Bellman equation*

$$J\left(i_t\right)=\min_{u_t=U} E\left\{g\left(i_t,i_{t+1}\right)+\alpha J\left(i_{i+1}\middle|i_t\right)\right\},\ \text{for } t=m-1,m-2,...,0, \tag{2}$$

where decision u_t is selected from a finite set of U at each time step, and the expectation operator is taken in respect to the probability in state transition from i_t to i_{t+1} by implementing decision u_t.

Despite the elegant equations of DP, difficulties in computation rise sharply as the complexity of the problem increases. To complete a single iteration of the algorithm, Eq.(2) needs to be computed for all i in the state space X. Furthermore, a policy has to be associated with a state and complete information of the underlying model is required to supervise state transition after implementing the policy. Therefore, application of DP approach is limited by three sources of constrains: dimensionality in state space, policy space and information space. This is referred by Bellman and Dreyfus (1959) as the “curse of dimensionality”.

The principle of ADP is to use approximation techniques to overcome the computational difficulties associated with DP, thus making it possible to apply sequential decision-making as exemplified by DP to complex problems. Corresponding to the source of dimensionality, ADP may approximate the value function, the policy or the underlying model. In this paper, we focus on value function approximation.

We use a continuous approximation function $\tilde{J}\left(\cdot,r\right): X\times\mathbb{R}^K\rightarrow\mathbb{R}$ to replace the exact value function $J\left(\cdot\right): X\rightarrow\mathbb{R}$, where parametric vector r of $\tilde{J}$ is K-dimensional. At each time step t, we calculate

$$\hat{J}\left(i_t\right)=\min_{u_t\in U}\underset{w_t}{E}\left\{g\left(i_t,i_{t+1}\right)+\alpha\tilde{J}\left(i_{t+1},r_t\right)\right\},\ \text{for } t=0,1,...,T-1 \tag{3}$$

and implement at each time step t

$$u_t^*=\arg\min_{u_t\in U}\underset{w_t}{E}\left\{g\left(i_t,i_{t+1}\right)+\alpha\tilde{J}\left(i_{t+1},r_t\right)\right\}. \tag{4}$$

The merit of using $\tilde{J}\left(\cdot,r\right)$ is that a look-up table of $J(.)$ values is replaced by a closed-form function. Supposing that there are n states and each may take m possible attributes, we reduce the dimension of state space from n^m to K, where K is the dimension of parametric vector r of $\tilde{J}\left(\cdot,r\right)$. This substantially reduces computational requirement, thus making solving complex problems online possible.

A popular approach to construct $\tilde{J}\left(\cdot,r\right)$ is to use *artificial neural networks* (ANN). This approach was firstly introduced by Werbos (1994) and then generalised by Bertsekas and

Tsitsiklis (1995) under the concept of *Neural Dynamic Programming.* In this approach vector *r* is represented by the neural weights connecting neurons in the network. Advantages of this approach include flexible function structure and availability of proven learning techniques. Disadvantage of this is that the black-box effects make it difficult to comprehend and generalise.

Alternatively, it is possible to identify a few *basis* functions $(\cdot)$ defined in state space $i \in X$ that capture the key features of the state, and form a linear separable approximation function as

$$\tilde{J}(i,r) = \sum_{n=1}^{N} \phi_n(i) r(n). \tag{5}$$

Once the basis functions are established, it is efficient to compute and simple to update *r*. In this work, we only investigate approximation techniques based on Eq.(5).

2.3. Reinforcement learning

The ADP approach is closely associated with reinforcement learning. This is because we usually do not know appropriate parametric structure and value of the approximation function *a priori*. A learning technique is required to supervise the evolution of *r* to guarantee that it converges to optimal value in static environment or reflex changing conditions in dynamic environment. Without ideal response for the learning agent to match, a learning agent has to learn from its own interaction with the environment and learn from "trail-and-error". This learning paradigm is denoted as reinforcement learning (Barto *et al*., 1983; Sutton and Barto, 1998).

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 shown as $g(\cdot)$, 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 to formulate a reinforcement learning problem. Typical reinforcement learning techniques include *Q-learning* (Watkins,1992) and *temporal-difference* (TD) learning (Sutton, 1988). The former requires a look-up table to present the set of (i, u), and therefore subject to the dimensionality of state space. The TD learning directly updates parametric vector of the approximation function, thus being more conforming to the task of overcoming computational difficulty.

The TD method constantly tracks the error between the estimated value and the observed value, and propagates the error signal back to the parametric structure so that

$$r^* = \arg\min_r \sum_{i \in S} \left[J(i) - \tilde{J}(i,r) \right]^2$$

Let the temporal difference d_t be defined as:

$$d_t = g(i_t, i_{t+1}) + \alpha \tilde{J}(i_{t+1}, r_t) - \tilde{J}(i_t, r_t). \tag{6}$$

For t = 0,1, …, the TD method updates r_t according to the formula

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

Applying (5) for approximation, we have

$$r_{t+1} = r_t + \eta_t d_t \sum_{k=0}^{t} (\alpha\lambda)^{t-k} \phi(i_k). \tag{7}$$

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

$$\sum_{t=0}^{\infty} \eta_t = \infty, \text{ and } \sum_{t=0}^{\infty} \eta_t^2 < \infty. \tag{8}$$

Parameter λ is known as *trace eligibility* factor, which takes value in [0,1]. Since temporal difference learning is actually a continuum of algorithm parameterized by λ, it is often referred as TD(λ). Furthermore, we may define a TD(λ) operator for $\lambda \in$ (0,1) by

$$\left(T^{(\lambda)} J\right)(i) = (1-\lambda) \sum_{m=0}^{\infty} \lambda^m E\left[\sum_{t=0}^{m} \alpha^t g(i_t, i_{t+1}) + \alpha^{m+1} J(i_{m+1}) \mid i_0 = i \right]. \tag{9}$$

In the case where λ=1, we have

$$\left(T^{(1)} J\right)(i) = E\left[\sum_{t=0}^{\infty} \alpha^t g(i_t, i_{t+1}) \mid i_0 = i \right] = J(i), \tag{10}$$

and for λ=0, we have

$$\left(T^{(0)} J\right)(i) = E\left[g(i_t, i_{t+1}) + \alpha J(i_{t+1} \mid i) \right]. \tag{11}$$

TD(1) is a true and unbiased estimation of $J(i)$, and 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. Tsitsiklis and Van Roy (1997) proved the convergence of r with TD(λ) algorithms for linear approximation function within the domain of infinite-horizon and finite-state discounted dynamic programming problems.

3. SYSTEM DYNAMICS AT AN ISOLATED TRAFFIC INTERSECTION

In this section we formulate system dynamics for an isolated traffic intersection. 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, …, *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,

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

The linear approximation function is formed by

$$\tilde{J}(i,r) = \sum_{n=1}^{N} \phi_n(i) r(n), \tag{12}$$

where

$$r(n) = \begin{bmatrix} r^-(n) \\ r^+(n) \end{bmatrix}. \tag{13}$$

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 *a*, where

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

We use vector *y* to 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

$$l_{t+1}(n) = l_t(n) - y_t(n) + a_t(n), \tag{14}$$

and signal vector *s* is transferred

$$s_{t+1}(n) = \left(s_t(n) + u_t(n)\right) \bmod_2, \tag{15}$$

where decision variable u_t takes

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

Equations (14) and (15) describe the state transitions of a single step. The number of steps in a certain time period depends on the resolution of the discrete time system. Let Δt denote 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 $M\Delta t$ seconds. In case where online information of vehicle arrivals a_t does not cover the *M* steps, we use Monte Carlo simulation to realise sample arrivals for the rest planning period.

For the *M*-step planning period, the ADP controller obtains

$$\mathbf{u}_t^* = \arg\min_{u_t \in U_t} E_{w_k}\left[\sum_{k=t}^{t+M-1} \alpha^{k-t} g\left(i_k, i_{k+1}\right) + \alpha^M \tilde{J}_{t-1}\left(i_{t+M-1}, r_{t-1}\right)\right],\ i \in X, \tag{16}$$

and calculates

$$\hat{J}_M\left(i_t\right) = \min_{u_t \in U_t} E_{w_k}\left[\sum_{k=t}^{t+M-1} \alpha^{k-t} g\left(i_k, i_{k+1}\right) + \alpha^M \tilde{J}_{t-1}\left(i_{t+M-1}, r_{t-1}\right)\right],\ i \in X, \tag{17}$$

Since we consider both vehicle delays and stops as performance indicator, the one-step cost function *g* is given by

$$g\left(i_t, i_{t+1}\right) = \sum_{n=1}^{N}\left[l_t\left(n\right) - y_t\left(n\right) + a_t\left(n\right)\right]\Delta t + Wz_t \,. \tag{18}$$

where z_t is the number of vehicle stops during Δt period, and W is the weighting factor. A greater W value shifts control priority to reduce vehicle stops, and vice versa.

The *M*-step temporal difference can be expressed as

$$\begin{aligned} d_M &= \hat{J}_M\left(i_t\right) - \tilde{J}\left(i_t, r_t\right) \\ &= \sum_{k=t}^{t+M-1} \alpha^{k-t} d_k\left(i_k, i_{k+1}\right), \end{aligned} \tag{19}$$

and the parameters are updated by

$$r_{t+1} = r_t + \eta_t \phi\left(i_t\right) \sum_{k=t}^{t+M-1} \alpha^{k-t} d_k\left(i_k, i_{k+1}\right),\ t = 0, 1, \ldots . \tag{20}$$

Equation (20) can be regarded as a special variant of (9). With a large *M*, Eq. (20) comes closer to (9) with λ = 1, and a smaller *M* makes (20) closer to (11) with λ = 0.

4. CONTROL POLICY

In distributed control architecture, each controller receives local traffic information and optimises performance according to local measurements. When a local controller plans ahead, it is assumed that controller status of other sites is kept the same. Traffic signals at an isolated intersection are grouped in to *phases*, which represent group of one or more traffic or pedestrian links that receive identical signal indications. In this study, intergreen is modelled as a phase. In this way, signal status moves immediately between any two phases. Given the current phase *p*, our controller decides a set of phases to simulate in the next *M* step. Figure 1 shows the possible paths to pick a set of phases to simulate. Terminal node 3 yields an empty set, which means the controller skips approximation steps because value approximation will be done in the next immediate following phase. Terminal nodes 4 and 5 mean that there are now two valid set of phases, corresponding to two valid options under the control policy: stay in this phase or move to next phase. Each terminal node represents the set of optional decision at any give time, which is represented by U_t. Optimal decision at any given time is obtained from (16), and implemented for a period of Δt.

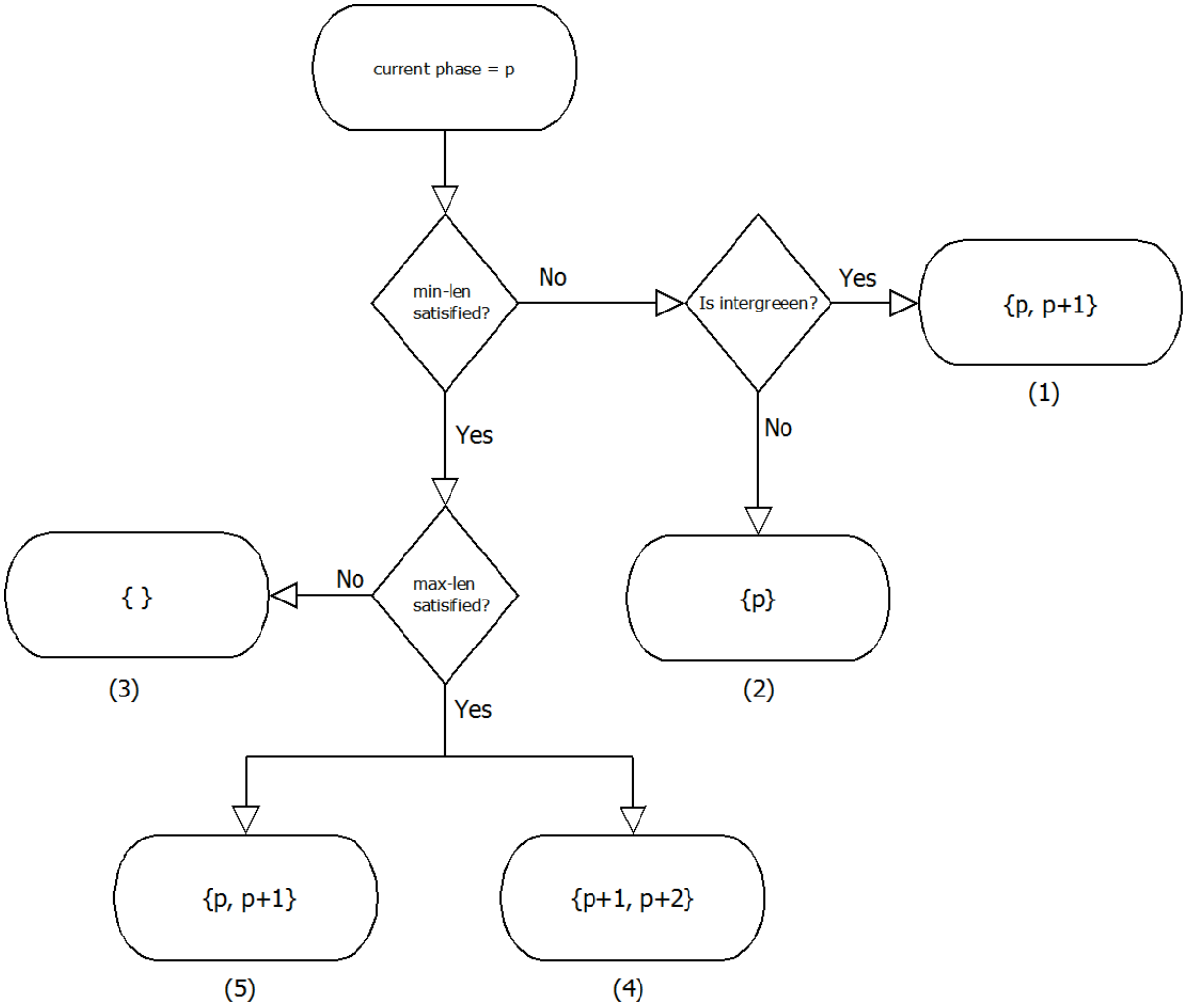


Figure 1: flow chart of deciding set of simulated phases

The traffic signal control algorithm using ADP can be summarised mathematically 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 a_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 $\mathbf{u}^*_t = \mathbf{0}$;
2.2 If signal change is admissible, for the planning period of *M*-steps, find the optimal decision $\mathbf{u}^*_t$ using (16).

Step 3: Update approximation function

3.1 Calculate new observation $\hat{J}_M(i_t)$ using (17)
3.2 Calculate current approximation $\tilde{J}_{t-1}(i_t, r_{t-1})$ using (12);
3.3 Calculate *M*-step temporal difference using (19)
3.4 Update functional parameter vector r_{t-1} using (20).

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

Step 5: Stopping Criteria

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

If the system is implemented in real world, T is set as infinity since the controller must be always on. In simulated environment, the value of T is preset; after that point, the simulation program terminates.

5. EXPERIMENT AND RESULT ANALYSIS

In this section we present the numerical experiments that implement the distributed ADP controller method to traffic network control, and provide result analysis. We begin with a few important assumptions used in the experiments in Section 5.1, and then discuss the configurations of numerical experiments in Section 5.2. Numerical results from a set of traffic network scenarios are present and analysed in Section 5.3, and the evolution of approximation function presented in Section 5.4.

5.1. Assumptions

We have the following assumptions for numerical experiments.

*Assumption 5.*1 Minimum greens and maximum reds*:*

Signal timings are subject to minimum green and maximum red time constraint. These constraints are translated to minimum length and maximum length of a phase.

*Assumption 5.*2 Positions of vehicles:

By putting a loop detector at the very upstream end of a link, we can record the time point a vehicle passes by. From that point, the current time, and maximum speed allowed on the road, we can calculate the approximate position of each vehicle on a link. We also need to have a detector at *D* meters upstream of the stop line to detect the presence of vehicle coming to the junction. The value of *D* is the maximum speed (measured in meters/second) allowed. For example, in our network, the maximum speed is 22.5 meters/second, so *D* equals 22.5.

5.2. Experiment configuration

5.2.1. Traffic model

The controllers are tested on a modified version of the Green Light District simulator (Wiering, van Veenen, & Koopan, 2004). The vehicle movement model used in this simulator is a cellular-automata model (Nagel & Schreckenberg, 1992). To make the model more realistic, we employed a higher resolution, in which cell size is 1.5 meters. The time step is 1 second per increment. With this configuration, each vehicle occupies 5 cells, and the maximum increase in speed is 2 cells/second. Let the velocity of a vehicle be *v*, its maximum velocity is v_{max} and the safety distance to the next vehicle ahead is *d*. The velocity update is done through this procedure:

a) Acceleration: $v \leftarrow \max\{\min\{v_{\max}, v+1\}, \min\{v_{\max}, v+2\}\}$;

b) Slowing down (avoid collision): $v \leftarrow \min\{v, d\}$;

c) Randomisation: with probability *P*, $v \leftarrow \max\{v-1, 0\}$.

5.2.2. Network topologies

We test our controller on 2 different configurations of network. In both networks, each road is two-way and has one link on each side. All vehicles go straight ahead at all junctions. The maximum speed allowed on all roads is chosen as 81 km/h in order to make it more precisely discretised into cells per second on the simulator. The value is translated into 15 cells per second on simulator. The distance from each origin to its closest junction is 500 meters. The link length between the 2 junctions is let varied between 200 meters and 330 meters. The 200-meter case is chosen because it is common in urban road networks. The reasons for choosing the 330-meters case are later expanded with discussion on signal settings.

In the first network, 2 junctions are lined up on an arterial road, as shown in Figure 2. The second road network is an extension of the first one, with 6 junctions lined up on an arterial road, as shown in Figure 3.

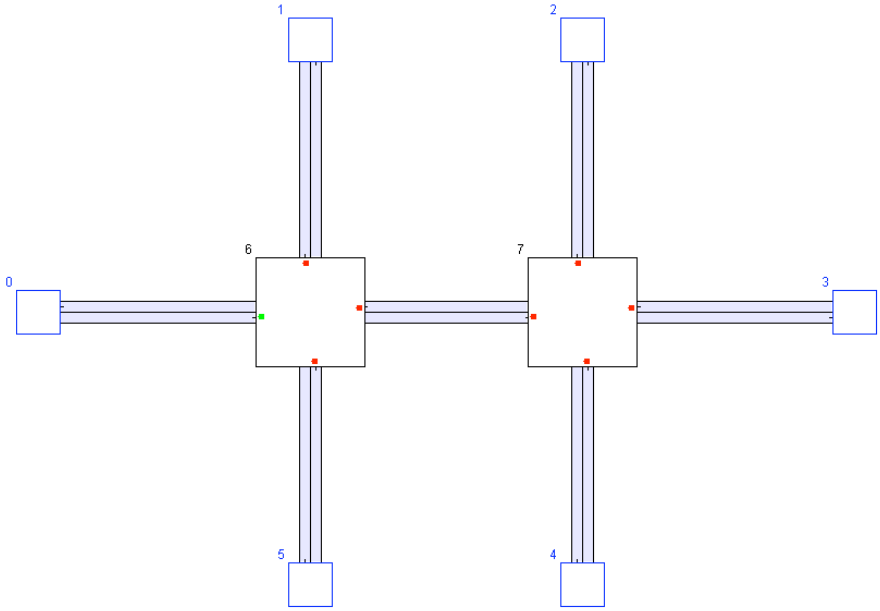


Figure 2: Simple network of 2 junctions at node-6 and node-7, other nodes are input sources

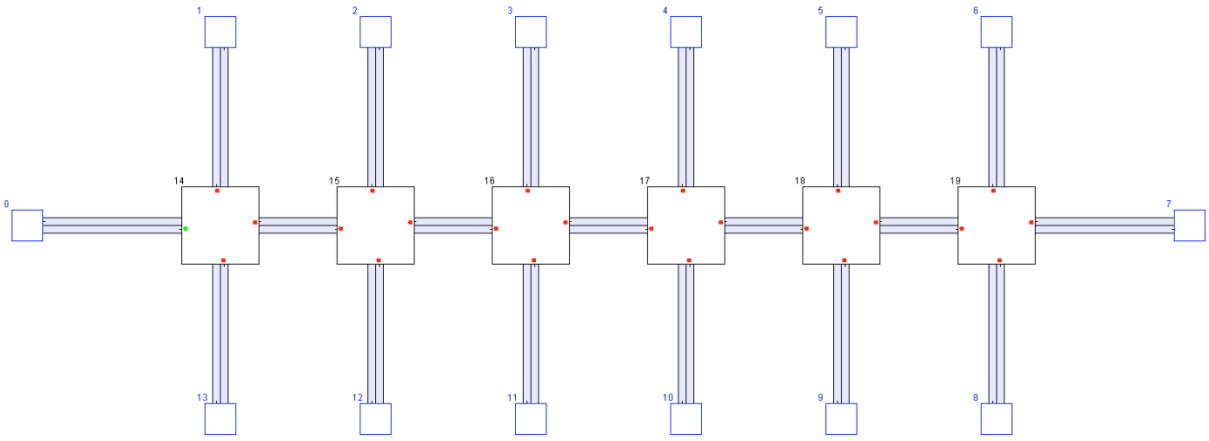


Figure 3: Network of 6 junctions at nodes 14 to 19, other nodes are input sources and absorbing places

5.2.3. Traffic demand scenarios

Vehicles are released into the system over 1 hour of simulation time with uniform probability, and the simulation terminates when all vehicles arrive at their destinations. The demand on all North-South links is 300 vehicles/hour, and is kept fixed during the simulation period in any scenario. The demand on the arterial (named it main road) varies in scenarios. There are three demand scenarios in total. In the first two scenarios, demand on the arterial links is assigned a value and kept fixed during the whole simulation period (1hour). Those values are 500 vehicles/hour (medium demand), 900 vehicles/hour (peak demand). The first two scenarios for the 2-intersection road network are summarised in Table 1, and that for the 6-intersection network in Table2. In the third scenario, the simulation time is still an hour but the demand varies every 20 minutes. It starts from 500 vehicles/hour, peaks at 900 vehicles/hour, and then reverts back to pre-peak flow. Using time-dependent demand, we

can benchmark our controller's performance with base-line controller in both fixed-demand scenarios and time-dependent scenarios.

Scenario name	Side road demand (vehicle/hour/origin)	Main road demand (vehicle/hour/origin)	Total demand (vehicle/hour)
Moderate	300	500	2,200
Heavy	300	900	3,000

Table 1: traffic demand scenarios for the 2-intersection network

Scenario name	Side road demand (vehicle/hour/origin)	Main road demand (vehicle/hour/origin)	Total demand (vehicle/hour)
Moderate	300	500	4,600
Heavy	300	900	5,400

Table 2: traffic demand scenarios for 6-intersection network

5.2.4. Traffic signal settings

For local intersection, signal schedule consists of 4 phases: North-South, intergreen, East-West, integereen. Each phase, except for intergreen phase, has a minimum length of 10 seconds and a maximum length of 80 seconds. Intergreen phase has an exact length of 5 seconds. With this signal setting, the minimum period between two consecutive East-West phases is 15 seconds, which we denote as "clear time". Link length 330 meters is chosen to exploit this "clear time" because the travel time (at aforementioned speed) for the link is approximately 15 seconds. With the aforementioned configurations of the road network, we explore the effect of each factor: traffic demand, link length, and the size of network. The tree in Figure 4 shows the detailed configurations that are tested. Each configuration is simulated 20 times.

5.2.5. Benchmarking method: TRANSYT

TRANSYT is an offline controller, so in the scenarios of fixed demands, it offers good benchmark for performance comparison. To make performance comparison consistent, we manually transfer TRANSYT plans to our simulator. The version in use is TRANSYT 12.0.

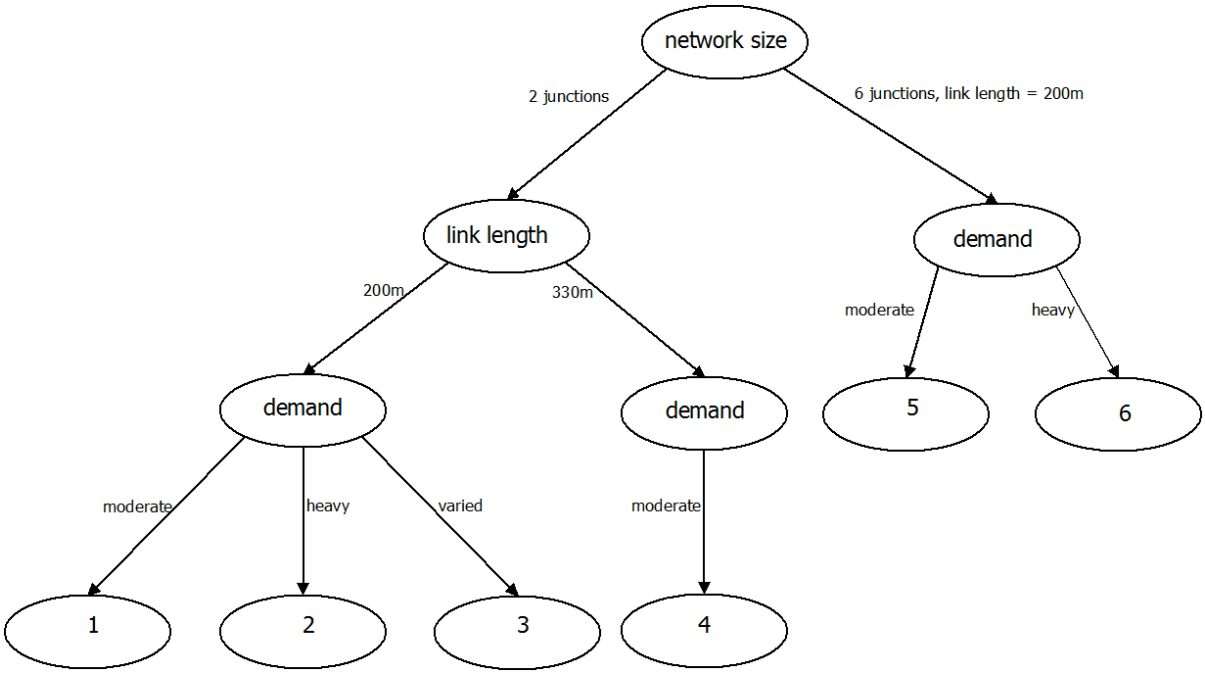


Figure 4: test scenarios, each leaf node is a scenario

5.3. Experimental results

Numerical results from applying moderate traffic demand to the 2-intersection network are shown in tables 3-5. Throughout the experiments, the value of *W*, the weighting factor for vehicle stops, is set at 20 according to (Sims, 1989).

5.3.1. Moderate demand

Link length	TRANSYT	ADP	Percentage of improvement
200 meters (scen. 1)	15.23	12.71	16.54%
330 meters (scen. 1)	14.18	8.25	41.78%

TABLE 3: average delay (seconds/vehicle), traffic demand scenario 1, 2-intersection network

Link length	TRANSYT	ADP	Percentage of improvement
200 meters (scen. 1)	0.96	1.01	-5.91%
330 meters (scen. 1)	1.03	0.69	33.12%

Table 4: average stops, traffic demand scenario 1, 2-intersection network

Link length	TRANSYT	ADP	Percentage of improvement
200 meters (scen. 1)	34.34	32.95	4.05%
330 meters (scen. 1)	34.84	22.07	36.65%

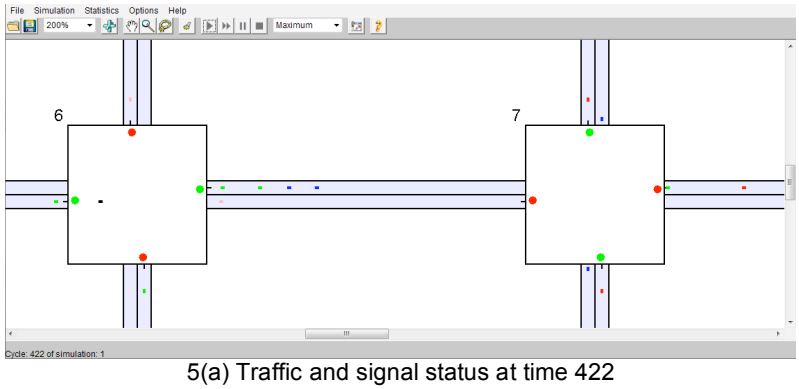
Table 5: average objective value (with W = 20), traffic demand scenario 1, 2-intersection network

As Table 5 shows, the ADP approach improved performance significantly from the TRANSYT plans in both cases of 200 and 330-meter. Both controllers work better when link travel time equals “clear time” than otherwise. In 200-meter case, ADP controller trade stop for delay to achieve a lower objective value. In 330-meter case, both methods discharged vehicles in platoons, thus producing better results than in the 200-metre case. However, TRANSYT optimised just one junction, sacrificing the other. When the platoon from node-6 reached node-7, corresponding traffic lights at node-7 turned green, producing a green wave, but when the platoon from node-7 reaches node-6, it was stopped for a few seconds. Although ADP controller did not require centralised policies, coordination was established voluntarily between adjacent intersections. Voluntary coordination becomes possible when vehicles released into the downstream link are detected by downstream sensors, and downstream controller plans ahead according to detected information. An illustrated example of voluntary coordination between adjacent intersections is shown in Figure 5.

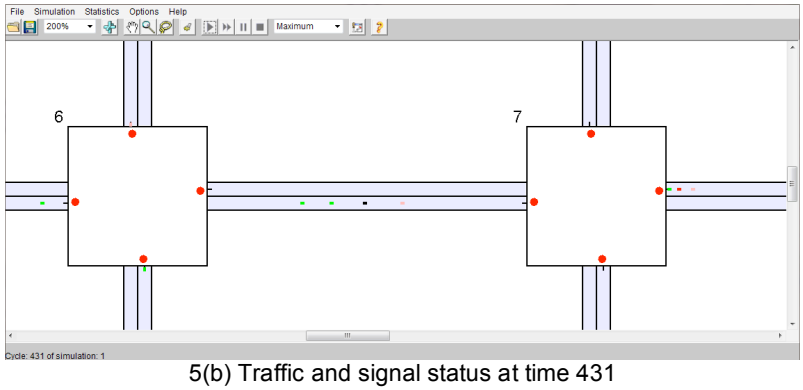
In the 200-meter case, link travel time is around 9 seconds, much shorter than the “clear time” 15 seconds. As the result, it was impossible to optimise offset in terms of platoon dispersion. In such case, ADP controller extended the East-West phase and only gave red light when the queue on North-South directions piled up to approximately 10 vehicles. In the same process, TRANSYT plans stayed fixed, and consequently yielded poorer performance. Table 6 shows average duration of East-West phases yielded by ADP and TRANSYT respectively.

Link length	Phase duration produced by TRANSYT (seconds)		Phase duration produced by ADP (seconds)
	Node-6	Node-7	
200 meters (scen. 1)	16	15	20.2
330 meters (scen. 1)	16	15	15.2

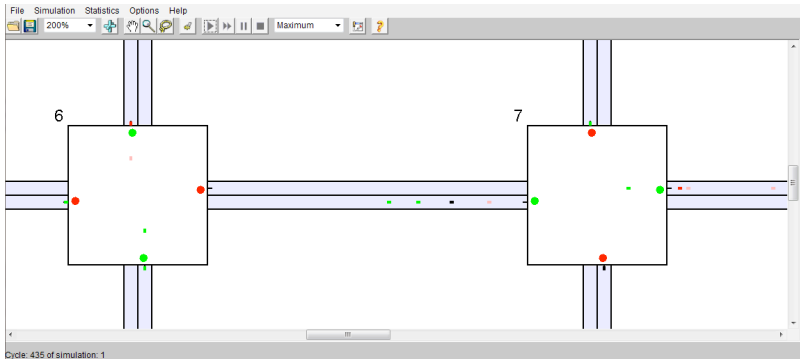
Table 6: Length of East-West phase (seconds), in moderate traffic, 2-intersection network



5(a) Traffic and signal status at time 422



5(b) Traffic and signal status at time 431



5(c) Traffic and signal status at time 435

Figure 5 An illustration of voluntary coordination between adjacent intersections controlled by distributed ADP controllers; 5(a): Inter-6 discharges vehicles while accommodating arriving platoon from inter-7; 5(b): Inter-6 changes signal after platoon arrival, departed vehicles travelling in link 6-7, inter-7 changes signal; 5(c): Iner-7 switches green to W-E phase just before the arrival of platoon.

5.3.2. Heavy traffic

For this scenario, we choose to test on the 200-meter case due to the small margin between performance of ADP and TRANSYT in the previous case. Tables 7-9 show the performance of two controllers in terms of averaged delay, averaged stops and averaged objective value respectively. In this scenario, the overall performance of ADP is still better than TRANSYT, with a wider margin than in previous scenario where traffic was light. ADP's gain is attributed to the reduced vehicle stops in comparison with the TRANSYT plans. On the other hand, the margin in reduced vehicle delays shrank. This understandable since the demand on East-West directions is high, and the best practice is to give longer green split to those, which can be confirmed by the indicators in Table 10.

Link length	TRANSYT	ADP	Percentage of improvement
200 meters (scen. 2)	16.8	15.67	6.76%

TABLE 7: average delay (seconds/vehicle), in heavy traffic, 2-intersection network

Link length	TRANSYT	ADP	Percentage of improvement
200 meters (scen. 2)	0.89	0.84	5.59%

Table 8: average stops (stops/vehicle), in heavy traffic, 2-intersection network

Link length	TRANSYT	ADP	Percentage of improvement
200 meters (scen. 2)	34.8	32.65	6.16%

Table 9: average objective value (with *K=20*), in heavy traffic, 2-intersection network

Traffic demand	Phase duration produced by TRANSYT (seconds)		Phase duration produced by ADP (seconds)
	Node-6	Node-7	
Moderate (scen. 1)	16	15	20.2
Heavy (scen. 2)	30	30	32.3

Table 10: Length of East-West phase (seconds) in scenarios 1 and 2, 2-intersection network

5.3.3. Time-dependent demand

To test the ADP's capabilities of adapting to variable traffic situations, a time-dependent demand profile is generated as the following. Demand on North-South origins is kept fixed as 300 vehicles per hour. As of the East-West origins, in the first 20 minutes, only 100 vehicles are released (which is equal to 300 vehicles/hour), in the next 20 minutes, 300 vehicles are released (which is equal to 900 vehicles/hour), and in the last 20 minutes, 100 vehicles are released. The demand profile is outlined in Table 11. In each 20-minute period, vehicles are released according uniform distribution. In this way, the total number of vehicles released in an hour from each East-West origin is 500 vehicles, which makes this scenario's net demand equal to that of moderate scenario. Two sets of TRANSYT plans were generated for peak and off-peak respectively. TRANSYT plans are appended one after the other in the order: off-peak, peak, off-peak. Performance comparison is shown in tables 12-14.

Origin	1st period (off-peak)	2nd period (peak period)	3rd period (off-peak)	Total number of vehicles
On N-S direction	100	100	100	300
On arterial links	100	300	100	500

Table 11: Demand profile for the time-dependent scenario.

Traffic demand	TRANSYT	ADP	Percentage of improvement
Varied (scen. 3)	17.01	15.1	11.22%

Table 12: average delay, time-dependent scenario, 2-intersection network.

Traffic demand	TRANSYT	ADP	Percentage of improvement
Varied (scen. 3)	0.98	0.71	27.65%

Table 13: average stops, time-dependent scenario, 2-intersection network.

Traffic demand	TRANSYT	ADP	Percentage of improvement
Varied (scen. 3)	36.7	29.3	20.03%

Table 14 Average objective value (with *W=20*), time-dependent scenario, 2-intersection network.

The ADP approach achieved 20% overall improvement from TRANSYT, with majority of the benefits coming from reduction in vehicle stops. The reduction in vehicle delay is also remarkable.

5.3.4. Large-scale network

On the large-scale network, the distance between any two intersections is 200 meters. Demand scenarios 1 and 2 were used in the experiments. In all scenarios, ADP controllers outperformed TRANSYT plans on a substantial scale, and particularly in the heavy demand case, as shown in tables 15-17. The TRANSYT schedules again created green waves for one direction at the expense of the opposite. In moderate traffic, it caused fewer stops than ADP does, as shown in Table 16. With heavier traffic, queue of the eastbound vehicles at Node-19 (Figure 3) spilled back to its upstream link and caused substantial degrade in performance (in terms of both delay and stops). Meanwhile, ADP controller broke down the platoon by using shorter cycles. As the result, vehicles were often let to have green wave at 2 intersections, then stopped at the following intersection.

Traffic demand	TRANSYT	ADP	Percentage of improvement
Moderate (scen. 1)	29.12	13.66	53.1%
Heavy (scen. 2)	94.63	29.25	69.09%

Table 15: average delay (seconds/vehicle) comparison, demand scenarios 1 and 2, 6-intersection network

Traffic demand	TRANSYT	ADP	Percentage of improvement
Moderate (scen. 1)	0.94	1.31	-39.44
Heavy (scen. 2)	3.65	1.38	62.02

Table 16: average stops (stops/vehicle) comparison, demand scenarios 1 and 2, 6-intersection network

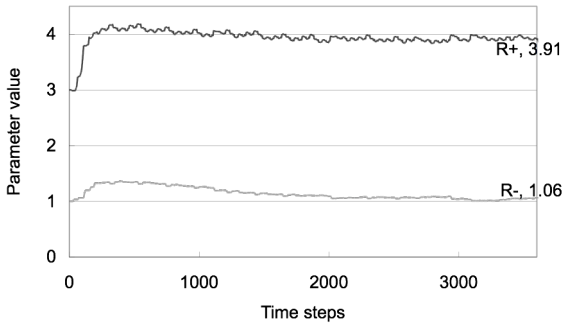
Traffic demand	TRANSYT	ADP	Percentage of improvement
Moderate (scen. 1)	48.01	39.99	16.7
Heavy (scen. 2)	167.64	56.98	66.01

Table 17: average objective value (with *W=20*) comparison, demand scenarios 1 and 2, 6-intersection network

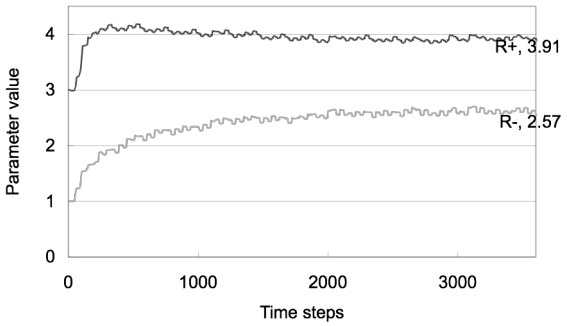
5.4. Evolutions of approximation

Approximation function (12) of any ADP controller was initialised with arbitrary values in all experiments. The arbitrary values are 1.0 for r^-, and 2.0 for r^+ for any traffic link. The update of r is governed by (20), which is computed upon very new observation of M-step state transition. Using TD learning, the parameters of (12) will converge to optimal value with probability of one so long as a few assumptions specified in Tsitsiklis and Van Roy (1997) are met. The ADP algorithm and the system dynamics formulated in this paper satisfy all of the assumptions, with the proof provided in Cai (2009). In the experiments, however, we use a constant stepsize η_t = 0.001, which does not satisfy the convergence assumption presented by (8). There are two reasons for this. The first is that diminishing stepsize rules like (8) assign most of weights to learning signals obtained in the first beginning of the simulation programme, where noise is strong and system is transient. This may result in over-shooting in parameter adjustment. Using a constant and cautious stepsize prevents over-shooting, but departs from eventual convergence. The second is that with time-dependent traffic, the system is dynamic and therefore will not arrive at steady-state.

Parameters in the case of static traffic with constant stepsize exhibit constraint oscillation around a statistical mean value, as shown in Figure 6.



6(a) Parametric values for Link65



6(b) Parametric values for Link67

Figure 6 Evolutions of parametric value of the approximation functions, 2-intersection network, first digit of link number as the destination of traffic and the second digit as origin, *r-* for green signal state, and *r+* for red signal state

6. CONCLUSION

In this paper we presented a study on applying approximate dynamic programming in distributed traffic network control. We proposed a closed-form linear function to approximate the exact value function of dynamic programming, thus reducing the computation requirement to a level manageable by a microprocessor of PC. The approximation is evolutionary and temporal-difference learning is used to provide learning signals. A cellular automation model is used to describe traffic dynamics, from which the controller withdraws information and constructs traffic state. The numerical results showed that in the 2-intersection network, the ADP approach improved control performance by 4 – 6 % from TRANSYT plans with time-invariant traffic and 20% with time-dependent traffic. In the case of larger network, the ADP approach improved performance by 16.7% with moderate traffic and 66% with heavy traffic from TRANSYT plans. The results highlight the readiness of the ADP approach to real-time urban traffic control, where traffic condition is constantly changing and degrees of saturation are usually high. Nevertheless, this study is one of the preliminary investigations in applying ADP to distributed network control. The approximation function is simple in structure and network configuration straightforward. Further studies from this on will investigate more accurate approximation functions, whose basis functions will capture the fundamental features of a more general representation of traffic network.

REFERENCE

Barto, A. G., Sutton, R. S., Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834-846.

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

Bellman, R., Dreyfus, S. 1959. Functional approximations and dynamic programming, *Mathematic Tables and Other Aids to Computation,* **13**(68), 247-251.

Bertsekas, D.P., Tsitsiklis, J.N., 1995. Neuro-Dynamic programming, Belmont, MA: Athenas Scientific.

Cai, C. 2007. An approximate dynamic programming strategy for responsive traffic signal control, *Proceedings of 2007 IEEE international Symposium on Approximate Dynamic Programming and Reinforcement Learning,* Hawaii, U.S., 303-310.

Cai, C. 2009. Adaptive traffic signal control using approximate dynamic programming, PhD thesis, University College London.

Cai, C., Wong, C.K., Heydecker, B.G. 2009. Adaptive traffic signal control using approximate dynamic programming, *Transportation Research Part C*, **17**(5), 456-474.

Gartner, N.H. 1982. Demand-responsive Decentralized Urban Traffic Control, Part 1: Single-intersection Policies, DOT/RSPA/DPB-50/81/24, U.S. Department of Transportation.

Gartner, N.H. 1983a. Demand-responsive Decentralized Urban Traffic Control, Part 2: Network Extensions, DOT/RSPA/P-34/85/009, U.S. Department of Transportation.

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

Henry, J.J., Farges, J.L., Tuffal, J., 1983. The PRODYN real time traffic algorithm, Proceedings of the 4th IFAC-IFIP-IFORS conference on Control in Transportation Systems, 307-311.

Heydecker, B.G., Cai, C., Wong, C.K., 2007. Adaptive dynamic control for road traffic signals, *Proceedings of 2007 IEEE International Conference on Networking, Sensing and Control*, London, United Kingdom, 193-198.

Hunt, P.B., Robertson, D.I., Bretherton, R.D., 1982. The SCOOT on-line traffic signal optimisation technique, *Traffic Engineering and Control*, **23**, 190-92.

Li, T., Zhao, D.B., Yi, J.Q. 2008. Adaptive dynamic programming for multi-intersections traffic signal intelligent control, *Proceedings of the 11th International IEEE Conference on Intelligent Transport Systems*, Beijing, China, 286-291.

Lowrie, P.R. (1992). SCATS – A Traffic Responsive Method of Controlling Urban Traffic.Roads and Traffic Authority, NSW, Australia.

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

Mirchandani, P., Head, L. 2001. RHODES: a real-time traffic signal control system: architecture, algorithms, and analysis, *Transportation Res. C*, **9**(6), 415-432.

Nagel, K., Schreckenberg, M. 1992. A cellular automaton model for freeway traffic. *J. Phys. I 2* , 2221-2229.

Sims, A. 1989. *SCATES user manual*. Sydney: Roads and Traffic Authority of New South Wales.

Sutton, R.S. 1988. Learning to predict by the method of temporal differences, *Machine Learning*, **3**, 9-44.

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

Teodorovic, D., Varadarajan, V., Popovic, J., Chinnaswamy, M.R., Ramaraj, S. 2006. Dynamic programming — neural network real-time traffic adaptive signal control algorithm, *Annals of Operations Research*, **143**, 123-131.

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.

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, U.K.

Watkins, C.J.C.H., Dayan, P. 1992. Q-learning, *Machine Learning*, **8**, 279-292.

Werbos, P.J. 1994. Approximate dynamic programming for real-time control and neural modeling, *Handbook of Intelligent control: Neural, Fuzzy, and Adaptive Approaches*, 493-515, Van Nostrand Reinhold, New York.

Wiering, M., van Veenen, J., & Koopan, A. 2004. *Intelligent Traffic Light Control*. Utrecht: UU-CS-2004-029.

Xiang, Y., Yi, J., Zhao, D. 2007. Multiple Approximate Dynamic Programming Controllers for Congestion Control, *Lecture Notes on Computer Science*, **4491**, 368-373.