ADAPTIVE TRAFFIC SIGNAL CONTROL

USING

APPROXIMATE DYNAMIC PROGRAMMING

by

Chen Cai

A thesis submitted for the degree of

Doctor of Philosophy

at the

UNIVERSITY COLLEGE LONDON

October 2009

Centre for Transport Studies

University College London

DECLARATION

For my parents, and for my love

ABSTRACT

This thesis presents a study on an adaptive traffic signal controller for real-time operation. An approximate dynamic programming (ADP) algorithm is developed for controlling traffic signals at isolated intersection and in distributed traffic networks. This approach is derived from the premise that classic dynamic programming is computationally difficult to solve, and approximation is the second-best option for establishing sequential decision-making for complex process. The proposed ADP algorithm substantially reduces computational burden by using a linear approximation function to replace the exact value function of dynamic programming solution. Machine-learning techniques are used to improve the approximation progressively. Not knowing the ideal response for the approximation to learn from, we use the paradigm of unsupervised learning, and reinforcement learning in particular. Temporal-difference learning and perturbation learning are investigated as appropriate candidates in the family of unsupervised learning. We find in computer simulation that the proposed method achieves substantial reduction in vehicle delays in comparison with optimised fixed-time plans, and is competitive against other adaptive methods in computational efficiency and effectiveness in managing varying traffic. Our results show that substantial benefits can be gained by increasing the frequency at which the signal plans are revised. The proposed ADP algorithm is in compliance with a range of discrete systems of resolution from 0.5 to 5 seconds per temporal step. This study demonstrates the readiness of the proposed approach for real-time operations at isolated intersections and the potentials for distributed network control.

ACKNOWLEDGEMENT

The work described in this thesis was carried out in the Centre for Transport Studies at University College London under the supervision of Professor Benjamin G. Heydecker. The author is most grateful to Professor Heydecker for his constant guidance and support. Many of the key developments in this doctoral study emerged from regular and friendly discussion with Professor Heydecker, and the reported work presents the major outcomes of our collaboration. The author is also grateful to secondary supervisor Dr. Francesca R. Medda for her help and guidance. The author would like to thank Professor Emeritus Richard E. Allsop for the help on applying scholarship and for the encouragement received.

The author thanks Dr. Chi Kwong Wong from University of Hong Kong for his direct participation in a number of studies that applied the methodologies presented in the thesis, and for the simulation programme for the traffic model used in multi-intersection control.

Professor Warren B. Powell gave the first impression of approximate dynamic programming and provided the introductory materials.

The author is thankful to Dr. Andy H.F. Chow, Abdul Memon, and Dr. Sanjay Rana for the discussion shared with them and support received. The friendships with Kamal Achuthan, Aris Christodoulou, Patricia Galilea, Qianwen Liu, Marcel Rommerts, Wei Yao and the rest of PhD student community at Centre for Transport Studies was the constant source of support and joy throughout the years of study.

The author thanks Dr. Nick Hounsell and Dr. Katerina Papadaki as examiners to this thesis. Their comments are valuable guidance for finalising the thesis.

Thanks are due to Rees Jefferys' Road Fund for supporting the work financially.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

The last three decades saw steady growth in car ownership and road traffic worldwide. In the United Kingdom, the Department of Transport (2009) reported that total road traffic increased by 87 per cent between 1980 and 2007, from 277 to 517 billion vehicle kilometres per annum. The majority of the growth has been in car traffic, which has risen by 88 per cent since 1980, from 215 to 404 billion vehicle kilometres. Trends with similar magnitude were seen in other major industrial countries as well as in emerging economies.

Rising road traffic intensifies the degree of congestion in road network, which in result causes prolonged travel time to the general public, adds extra cost to economic activities, and raises the pressure on road safety and environment. Congestion effect in the U.K., using the current government method of evaluation, is that the annual cost of £20 billion would increase to £30 billion by 2010 (Goodwin, 2004). In the USA, traffic congestion caused $78 billion annual drain on economy in the form of 4.2 billion lost hours and 11 billion litres of wasted fuel (Texas Transportation Institute, 2007). Congestion cost reached 267 billion Euro per annum for the EU-17 in 2000 (INFRAS/IWW, 2004).

Managing road congestion, therefore, is of strategic value to the pursuit of sustainable activity and economic development. For this end, cities, regional councils, and state transport agencies are persistently searching for ways to mitigate urban traffic congestion, while minimising costs and maintenance requirements. There are several ways to tackle road congestion. On the macro-level, it is common to use economic levers to regulate traffic demand, and encourage transport mode switches to benefit strategic interests. Examples of this are fuel taxes and congestion charges to private vehicles. At the micro-level, intersections of urban areas often limit network capacity and are common congestion points. Therefore, controlling traffic that has already entered into urban road networks relies on having an efficient and well-managed traffic signal control systems.

1.1 Traffic Signals

Traffic signals are used to manage conflicting requirements for the use of road space by allocating right of way to different sets of mutually compatible traffic movements during distinct time intervals. The objectives of signal control vary in accordance with the prevailing policy of urban traffic management and control.

Although the history of traffic signals dates back to 1914 in the USA, their operation acquired prominence in the post-war era, since when road networks have become increasingly congested. The evolution of traffic signal control concept saw broadly three generations. The first-generation had preset signal sequence and duration, and required manual maintenance. This kind is usually referred as fixed-time methods. The Traffic Network Study Tool (TRANSYT, Robertson, 1969; Vincent *et al.*, 1980) is one of the established tools for calculating fixed-time plans. The second-generation systems, which largely came into service in the 1980s, are characterised by the feature of adjusting signal timings according to detected traffic at real-time. Inductive loops are commonly used, and microprocessors facilitate real-time process of information and calculation of signal timings. Successful commercial products of this sort are the Split Cycle Offset Optimisation Technique (SCOOT, Hunt *et al.*, 1982) and the Sydney Coordinated Adaptive Traffic System (SCATS, Luk, 1984). Each of those products has been employed in more than one hundred cities worldwide. In field evaluations, the optimised responsive systems such as SCOOT consistently outperform previous method, as shown by the results in Table 1-1. The benefits from responsive systems, together with rapid advances in communication and information technologies, have driven the development of a new generation of concept of signal control system. The third-generation is distinguished by dynamic decision-making and distributed control structure. This generation of system is fully adaptive, i.e. with adjustable control parameters and adjusting routines, and signal timings are optimised progressively over time as detector information becomes available. The quantities to be calculated are the sequences of signal changes and the associated timings. These decisions are based upon estimates of current queue lengths and information from detectors about traffic which will arrive at the intersection within the next few seconds, as well as about traffic that is leaving the junction. For network control, these systems exploit the computational power of standalone microprocessors to operate at separate local sites. Although prototypes of this sort emerged as early as in the 1980s, such as OPAC (Gartner, 1982, 1983a, 1983b), PRODYN (Henry *et al*., 1983), UPTOPIA (Mauro et al., 1989) and RHODES (Mirchandani and Head, 2001), this generation of controller remains largely in the stage of development or in field evaluation.

A full review of both established and developing control methods is provided in Chapter 2 of this thesis.

Table 1-1 SCOOT Field Evaluation Results

Location of SCOOT Installation	**Previous Control Method**	**Year**	**% Benefit over previous control method**	
			Delay	**Travel Time**
São Paulo, Brazil (ver. 2.4)	Fixed-time (TRANSYT)	1997	0 - 40	-
São Paulo, Brazil (ver. 3.1)	Fixed-time (TRANSYT)	1997	0 - 53	-
Nijmegen, The Netherlands (ver. 2.4)	Fixed-time	1997	25	11
Toronto, Canada (ver. 2.4)	Fixed-time	1993	17	8
Beijing, China (ver. 2.3)	Fixed-time (Uncoordinated)	1989	15 - 41	2 - 16
Worcester, UK (ver. N/A)	Fixed-time (TRANSYT)	1986	3 - 11	7 - 18
Worcester, UK (ver. N/A)	Isolated Vehicle Actuation	1986	7 - 18	15 - 32
London, UK (ver. N/A)	Fixed-time	1985	19	6 - 8
Southampton, UK (ver. N/A)	Fixed-time	1985	39 - 48	18 - 26
Coventry, UK - Foleshill Road (ver. N/A)	Fixed-time (TRANSYT)	1981	22 - 33	4 - 8
Coventry, UK - Spon End (ver. N/A)	Fixed-time (TRANSYT)	1981	0 - 8	0 - 3

Source: Mountain-Plains Consortium (MPC) Report No. 03-141 Adaptive Signal Control II (2003)

1.2 Control Methods for Traffic Signal

Controlling traffic signals at intersections is a challenge that has both theoretical and practical value. Control variables for traffic signals usually include *cycle time*, *green split*, and *offset*. Cycle time is the duration of a repeatable signal timing sequence, and green split determines allocation of cycle time to competing road users. Offset regulates the coordination between adjacent intersections.

Control methods can be broadly divided into non-optimised and optimised categories. The non-optimised methods use a set of heuristic rules to define relationships between signal timings and traffic conditions. System D (Department of Transport, 1984) and SCATS are examples of this sort. The optimised methods calculate timings to satisfy control objectives that usually aim to minimise estimated vehicle delays and stops, equalise degree of saturation on approaching links, or maximise intersection capacity.

The optimised methods can be further divided into off-line and on-line groups. Early research in optimisation calculated fixed-time plans off-line. For isolated intersections, Webster (1957) used unconstrained optimisation to minimise approximated average vehicle delay in respect of cyclic length and green split. Allsop (1971a, 1971b) used linear and convex programming to solve constrained formulation that aims to maximise reserve capacity and to minimise average rate of delay. As for optimised network control, Little (1964) used mixed-integer-programming (MILP) to find the solution to coordination between a pair of intersections. The TRANSYT system employs a cyclic traffic model to facilitate a direct search technique that minimises total rate of delay and stops in a network.

An example of optimised on-line system is the SCOOT system, which employs optimisation routines for cycle, split and offset respectively. The optimisation routines are limited to choice of incremental changes in signal timings. Cyclic traffic profiles are used to evaluate performance measures.

Research in advanced adaptive systems that lead to the development of OPAC, PRODYN, and RHODES uniformly recognised the importance of dynamic programming (Bellman, 1957) in solving sequential decision-making for complex systems. Dynamic programming (DP) decomposes a complex problem into a series of sub-problems with discrete time *steps* between them. At each time step, the system is characterised by a number of *state* variables that specify the sub-problem. The more complex the sub-problem is, the greater the size of the *state-space*, which is a *n*-dimensional space whose axes are the state variables. Consequently more calculation is required to solve each sub-problem. Computation routine of DP involves construction of the *value function* that associates expected future value with each state. This value function, which is central to DP, serves to evaluate possible actions in order to achieve optimality. The range of different possible actions is defined by a specific control *policy*. The routine of DP is iterative, and executing an action results in a transition from a state at one time to another state at a later time. A *model* of the system is used to represent the dynamics of state transition. Finally, the DP algorithm requires a distinct property referred as the *principle of optimality*, which says that knowledge of the current state of the system conveys all the information about its pervious behaviour necessary for determining the optimal policy henceforth. Any problem lacking this property cannot be formulated as a DP problem. Under this principle, results obtained from DP are global optimal.

However, the advantages of DP are countered by difficulties in implementation. Difficulties come mainly from two factors. One is the computational requirement associated with the size of state space, and the other is the incompleteness of information. Intensive computation requirement suggests that DP is not compatible with real-time operation where possible actions have to be evaluated within a short time and computation power is limited at standalone facilities. Additionally, requiring complete traffic information concerning future arrivals is usually unrealistic. In fact, for traffic signal control, no exact DP algorithm has ever been implemented for an operational prototype.

Other approaches were therefore sought to achieve sequential decision-making that approximates the behaviour of DP. Artificial intelligence (AI) techniques are popular candidates for desk-top research. However, the closed black-box style commonly associated with AI solutions, such as artificial neural networks, genetic algorithms and fuzzy logic, make comprehension and generalisation difficult. Furthermore, from a practical point of view, dependability of the traffic signals is vital to road safety at intersections. Putting safety of road users into hands of inscrutable and hence incomprehensible controllers raises ethical issues of responsibility. On the other hand, using approximation techniques can reduce the difficulties associated with DP. A possibility that is practically related to the present case of signal control is to develop approximations for paths of the decision process that lie in the future. Approaches like this are referred as approximate dynamic programming (ADP).

1.3 Approximate Dynamic Programming

Approximation methods for DP can be broadly divided into a few categories according to what is being approximated. As was discussed in the previous section, key components of the DP routine are the model of the system, control policy and the value function. In this regard, we have three main categories for approximation.

1. Model approximation. This approach is used in cases where the dynamics of the system are too complicated to model, or are only partially observable. Complex dynamics can be approximated by a simpler model, thus reduce the complexity and computational demand. Examples of this in traffic control are macroscopic models that describe vehicle motions in simple terms. Taking Daganzo's Cell Transmission Model (1994) as an example, acceleration and deceleration are not modelled, and traffic either stops in a queue or moves forward at constant speed. Using this model, fewer state variables are required to describe the dynamics of the traffic than models that consider acceleration and deceleration of vehicles individually.
2. Policy approximation. This approach involves parameterisation that captures the relationship between control policy and independent variables. In traffic control, we may propose a parametric structure to relate extension of green time to length of queue (Teodorovic *et al.*, 2006). If this approach is successful, over iterations, value of parameters converges, and an optimal control policy is therefore established. The main drawback of this approach is that the *policy-evaluation* routine, which evaluates optimal control policy, can be intractable itself. In this case, one must resort to gradient-based methods that search for local optima of policy variables.
3. Value function approximation. This approach parameterises the value function and computes parameter values that lead to an accurate approximation to the optimal value

function. Desirable properties of this approximation structure include differentiability, reduced complexity compared to lookup-table representation, appropriate algorithms for computing parameter values, and capability of approximating the optimal shape of the function with accuracy (Ferrari and Stengel, 2004).

Approximation to the entities described above can be either performed separately or jointly. For example, Werbos (1994) proposed an algorithm that combines policy approximation and value function approximation. Nevertheless, value function approximation is central to the concept of ADP. Using this approach, algorithms for computing parameters can be variants of exact DP algorithms. This implies that an important family of machine learning can be incorporated for computing parameters online — this is known as *reinforcement learning*.

The ADP concept has important implication for several engineering fields. Studies of possible applications of ADP have been seen in intelligent electric power grids (Venayagamoorthy *et al.*, 2000), flight control (Ferrari and Stengel, 2004), space vehicle design (Kulkarni and Phan, 2003), large-scale logistic problems (Powell, 1996; Powell and Topaloglu, 2003) and resource allocation problems (Papadaki and Powell, 2002, 2003; Powell and Van Roy, 2004).

1.4 Reinforcement Learning

Reinforcement learning is used to estimate values for the parameters of specified functional relationships between actions and their effects. A system that is learning from a control process can be seen as a learning *agent*. The agent is not told which actions to take, but instead must discover which action yields the best performance by trying them. In the most interesting and challenging cases, actions may affect not only the immediate performance but also the next state and, through that, all subsequent performance. Two characteristics — trial-and-error, and delayed effects — are the main distinguishing features of reinforcement learning. This learning technique is closely related to DP, as it relies on the formulae of DP to estimate future values of performance as well as the immediate ones. In this regards, reinforcement learning fits well into the category of value function approximation.

There are many theoretical and practical issues associated with the use of reinforcement learning for value function approximation. The most important ones pertain to the convergence of parameters. Previous studies that presented convergence results include Sutton (1988), Watkins and Dayan (1992), and Tsitsiklis (1994), all of which consider only cases where the number of adjustable parameters is the same as the cardinality of the state space. The more general case, involving the use of function approximation, was addressed by Dayan (1992), Gordon (1995), Tsitsiklis and Van Roy (1996, 1997) and Singh *et al.* (1995), who established convergence with probability 1 to a linear approximation of the value function. General conditions for convergence using nonlinear approximation are yet to be established. An example of divergence using nonlinear approximation function and trained by reinforcement learning was presented in Tsitsiklis and Van Roy (1997).

1.5 Objectives of this Doctoral Study

This study aims to develop adaptive traffic signal control by applying state-of-the-art ADP techniques. The focuses of methodological development are to investigate value function approximation using reinforcement learning, and establish a theoretical framework in which various approximation structures and reinforcement learning techniques can be incorporated for the case of adaptive traffic signal control.

From the engineering aspect, this study aims to develop the ADP concept for distributed real-time traffic signal control, in which case standalone facilities are limited in computational power and information of future traffic arrivals emerges over time. Operation environments that can be accommodated by the method presented here include most isolated intersection layouts and also typical grid traffic networks. Computational demand of a single ADP controller should be sufficiently managed by an ordinary PC available in the current market. Online information is received from ordinary loop detectors. Communication between neighbouring intersections may use existing traffic management facilities.

1.6 Organisation of the Thesis

The reminder of the thesis is organised as follows. In Chapter 2, we review the established traffic signal control methods as well as relevant state-of-the-art concepts. In Chapter 3, a systematic investigation is presented for applying the ADP concept for real-time control. This chapter first identifies the limits of the DP, then introduces basic ideas of ADP and approximation structures, and finally discusses applicable machine learning techniques. In Chapter 4, we introduce the models of traffic dynamics at signalised intersection and establish a few important structural properties of the value function, after which an appropriate ADP structure and algorithm are proposed. Chapter 5 contains a series of numerical results from experimental scenarios that include both isolated intersections and small coordinated traffic networks. Concluding remarks are presented in Chapter 6.

CHAPTER 2 TRAFFIC SIGNAL CONTROL METHODS

This chapter provides reviews of established as well as developing methods for traffic signal control. The general objective for traffic signal control is recognised by Wood (1993) as:

> "Promote the objectives of urban traffic management and control in many different ways, including both tactical considerations and more strategic ones. The general purpose of tactical traffic management includes ensuring good operations of the junction and network with current and expected arrivals of traffic. The purpose of strategic traffic management is broader and includes possibilities such as prioritisation and promotion of different groups of travellers such as pedestrians or bus passengers by provision of appropriate facilities, and limitation of capacity for motor vehicles to manage traffic growth."

The objectives are managed by operating signals either locally or co-ordinately throughout network. Heydecker (2004) identified control decisions for signal controllers as:

1) The order in which signals are switched to green indications;

2) For how long each green indication should persist.

There is a rich literature on traffic signal control. This review focuses on those with distinguished implication to research and engineering. The rest of this chapter is organised as follows. Important terminologies for traffic signal control are introduced in Section 2.1. Established control methods are reviewed in Section 2.2. Developing methods in adaptive control, particularly those involve machine learning, are reviewed in Section 2.3. The research scope and methodologies for this doctoral study are discussed in Section 2.4.

2.1 Traffic Signal Terminologies

The definitions of key terminologies of traffic signals are as follows.

Link: A group of adjacent lanes on which traffic forms a single combined queue.

Phase: A group of one or more traffic or pedestrian links that receive identical signal indications.

Stage: A set of one or more traffic and/or pedestrian phases that receive a green signal during a particular period of the cycle.

Inter-green: The period between the end of the green display for one stage and the start of the green display for the next stage.

Minimum (Maximum) green: The minimum (maximum) permitted period of green display for a phase.

Cycle: Usually considered to be the time between successive starts of the green stage 1.

Offset: Offset is a time difference between the start times of two signal phases of stage 1 at adjacent intersections.

The relationships between phase, stage, inter-green and cycle are illustrated in Fig.2.1. An example of signal coordination with offset and a fixed-time plan is shown in Fig.2.2.

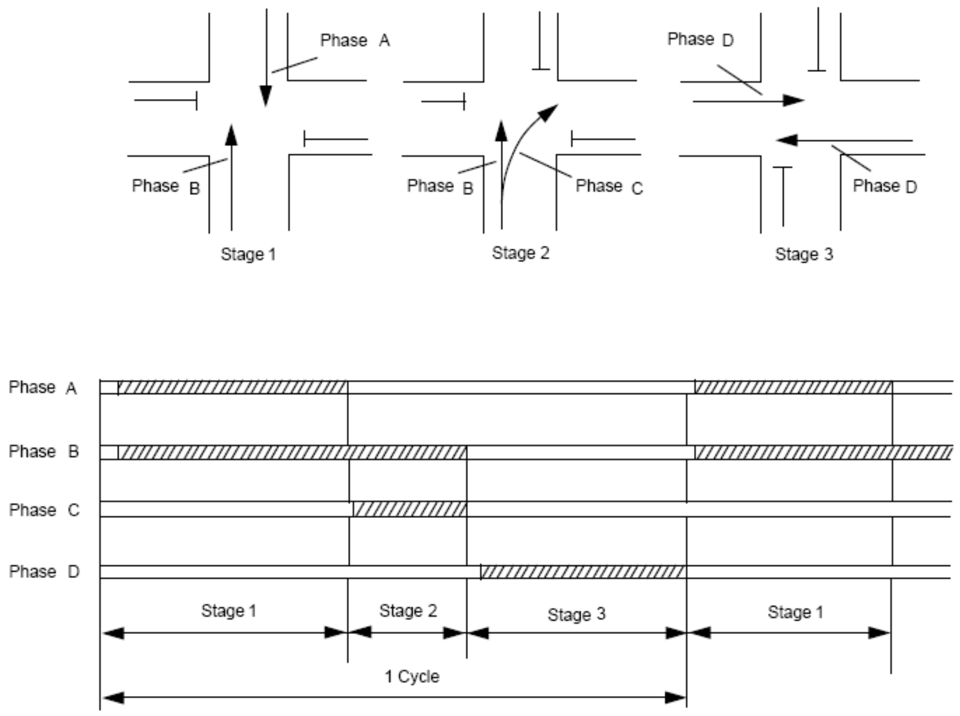


Fig. 2-1 Definitions of Phase, Stage and Cycle as traffic signal control terminologies, and their relationships; mutually compatible phases are grouped into stages, and certain phase may appear in more than one stages, such as Phase B in Stage 1 and Stage 2.

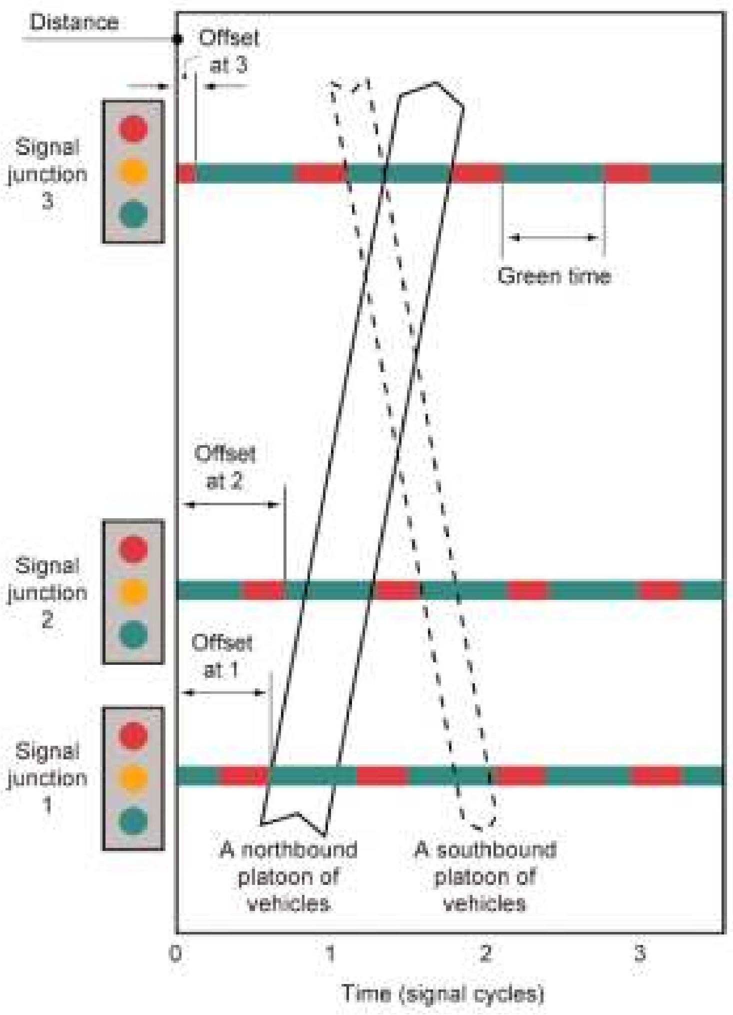


Fig. 2-2 An idealised time-distance diagram showing signal coordination with offsets and a fixed-time plan; both of the northbound and southbound movement pass a cascade of intersections without stop and reduction in speed. In this case offsets are measured from time 0 to the starting of common phase at each intersection. Source: Traffic Advisory Leaflet 1/06, Department of Transport, U.K.

The operation of traffic signal settings can be classified broadly into fixed-time and traffic-responsive. Fixed-time methods use historical traffic data to calculate stage sequence, duration and the associated inter-stage structure in advance. By contrast, traffic-responsive methods adjust indicated green times according to observed traffic flows. Traffic detectors, loop detector in particular, are common instruments for observing road traffic at real-time.

These are two distinct styles of formulation for control decisions. The first one is *stage-based*, in which signal controller determines the sequence and duration of stages. This has the advantage of dividing time into a series of intervals throughout each of which a single stage runs; these intervals are separated by inter-stage periods within which signals change between green and red. The second style is *phase-based*, in which controller determines the sequence and duration of phases in a cycle. Control decisions can be described within each of these two styles, and hence optimisation formulations can be either stage-based or phase-based.

2.2 Established Control Methods

In this section we review established control methods, including fixed-time and traffic-responsive methods. The established methods are well acknowledged in traffic engineering as well as in analytical studies.

2.2.1 Fixed-time methods

Fixed-time control methods are usually calculated under the assumption that for some specified time intervals the mean rate at which traffic arrives is constant and usually is within the range that can be accommodated at the junction. Short-term variations in arrival rates are admitted as random variations in the arrival processes and are ignored by the control strategy.

Webster (1957) considered minimising the sum of flow-weighted delays at a junction. Webster's method is stage-based, and assumes a fixed cyclic sequence of stages and inter-stage structures. By considering only the most heavily loaded links, Webster devised rules to calculate stage durations and a cycle time that approximately minimise the rate of delay as estimated by his own formula. These rules were derived using an approximate analysis of a simple junction configuration and can only be applied when the sequence of stages is sufficiently simple.

Allsop (1971a) formulated delay minimisation and capacity maximisation based on the stage-based approach. Like Webster's method, this requires pre-specified stage sequence and inter-stage structures. More flexible signal timings allocation can be achieved if individual phases are taken into consideration in the design. Without the need to maintain specific stage structure, signal timings can be assigned directly to individual phases as long as mutually incompatible phases are separated by sufficient inter-green times for safe operation.

The phase-based formulae are more flexible than the stage-based one because the sequence of signal indications and the structure of the inter-stage periods are implicit endogenous variables. Heydecker and Dudgeon (1987) showed that, in case of more complicated junction layouts, considerable benefits could be gained by using the starting times and durations of green signal indications for phases as variables rather than working through the intermediary of stages. This phase-based formulation also offers a convenient means to represent interactions between traffic links. However, the benefits are achieved at the expense of requiring a greater number of variables and constraints. Improta and Cantarella (1984) formulated the phase-based approach for the design of traffic signals as a Binary-Mixed-Integer-Linear-Program (BMILP). More recently, the lane permitted movements have also been represented as binary variables to form an extension to the phase-based design framework which supports the integrated design of junction geometry, lane allocation and signal timings (Wong and Wong, 2003). It is expected that junction layouts once determined will not be reconstructed during daily operations. Nevertheless, signal timings can be fine-tuned to accommodate latest demand patterns. Off-line methods of this kind require information about approaching flows for their calculation. Signal settings can usually be optimised analytically for practical implementation.

Examples of established software packages for fixed-time signal plans include the Traffic Network Study Tool (TRANSYT, Vincent *et al*., 1980), Optimised Signal Capacity and Delay (OSCADY). The plans obtained from the software packages usually serve to provide a benchmark for analytical studies and a reference library for operation.

2.2.2 Non-optimised traffic-responsive methods

Non-optimised systems use a set of heuristic rules to relate signal timings to detected traffic conditions. Real-time adjustments to signal timings are not optimised in respect to performance measures.

System D or vehicle actuated (Department of Transport, 1984) is implemented by using vehicle detectors to estimate when the flow-rate over the stop-line falls below the saturation level, as would occur when a queue is dissipated. In practice, the output from detectors on different links is pooled and a stage will be extended until no vehicles are detected during a critical interval on any link that will lose right of way.

Van Zuylen (1976) developed a method that uses similar heuristic within a phase-based framework. This incorporates additional flexibility as no sequence need be specified. Then, the controller is free to change green indication from phases having no queues remaining and to select other phases according to their demands and their compatibility with phases still running. Although this method is relatively easy to implement, the more complex an intersection, the further it departs from the simple two-link case for which the policy was developed. The non-uniform arrivals, which occur more often in reality, will further reduce the validity of the policy.

Sydney Coordinated Adaptive Traffic System (SCATS, Luk, 1984) is a two-level hierarchical system developed in Australia by the Road and Traffic Authority (RTA) of the state of New South Wales. The SCATS uses information from vehicle detectors, located in each lane immediately in advance of the stop-line, to adjust signal timings. The SCATS does not optimise signal timings in respect to performance measures. Instead, it acts as a heuristic feedback system adjusting signal timings based on changes in traffic flows from previous cycles. The system assumes that higher cycle lengths mean greater intersections capacity, and therefore calculate splits proportional to approach demand and longer offsets for increased traffic volumes. Measurement for demand is degree of saturation (DS) and that for traffic volume Link Flows (LF). A linear relationship between DS and cycle length is assumed. For a given LF pattern, offset is adjusted according to a linear relationship with cycle length. Signal timings are incrementally adjusted every cycle to avoid large fluctuations.

Non-optimised systems mechanically match detected traffic conditions to preset heuristic rules. Although these systems are simple to implement and robust in control, control performance is not optimised.

2.2.3 Optimised traffic-responsive methods

Optimised traffic-responsive methods usually involve state-space representation of control system and sequential decision-making. The control objectives are commonly set to optimise some measures of generalised control performance over a time period, whilst accommodating both systematic and random variations. The quantities to be calculated are the sequences of signal changes to be invoked and the associated timings. This can be formulated in dynamic programming (DP), and solved by using Bellman's equation (1957). Important to DP formulation of the control problem are definitions of state variable. Bell *et al.* (1990) identified the state of a traffic signal control system as a composite of two elements: the state of traffic and the state of controller. They further elaborated that:

> "The state of traffic at a junction can be specified by the number of vehicles queuing in each of the links and the arrivals of vehicles in the near future: the former of these is influenced by the signal controls applied. The state of the controller can be specified by the signals that are green, any changes that are currently underway, the times at which they will be completed and the times of expiry of any minimum or maximum permitted durations."

This structure of state imposes a substantial computational difficulty for implementing Bellman's equation, because in general the state space to be investigated corresponds to the product of all possibilities for each of these state variables. In addition to the computational difficulty, Bell *et al.* further commented on DP solution for traffic signals:

> "Normal backward dynamic programming techniques are not particularly suitable for use in real-time control of this kind. This is because an unnecessarily large number of state sequences are considered and calculations commence at the end of the look-ahead where information on arrivals is least certain."

This comment reaffirms conclusions drawn in Robertson and Bretherton (1974), Gartner (1982) and Henry *et al.* (1983).

DYPIC (Robertson and Bretherton, 1974) is a DP approach that serves only for analytical purpose. The computational difficulty of DP solution restricts implementation of DYPIC for engineering purpose. The authors proposed a quadratic function to approximate exact value function, and developed a heuristic solution based approximation function. The heuristic solution adopts the concept of *rolling horizon*, which means that: first, the planning horizon is split into a 'head' period with detected traffic information and a 'tail' period with predicted traffic information; second, an optimal policy is calculated for the entire horizon, but is only implemented for the 'head' period; finally, when the 'head' period expires and new information becomes available, the process rolls forward and repeats itself.

OPAC (Gartner, 1982, 1983a, 1983b) is a distributed real-time traffic signal control system. OPAC does not use the formulae of dynamic programming; rather it uses optimal sequential constrained search (OSCO) to plan for the entire horizon, and employ terminal cost to penalise queues remaining in the system at the horizon. The horizon is 60 seconds (60s) long, 10s of which is the 'head' period supplied with detected real-time traffic information, and the rest with predicted traffic information. Gartner (1983a) reported that OPAC in both simulation and field tests saved 5-15% from existing traffic-actuated methods, with most of the benefits coming from situation of high degree of saturation. The concerns of OPAC are that the restrictions in OSCO search reduces the flexibility of decision making, and a long planning horizon (60s) raises practical questions about optimising far into the future on the basis of predicted information when the decisions planned for the 'tail' may never the implemented.

PRODYN (Henry *et al.*, 1983) adopts rolling horizon approach and extends Robertson and Bretherton (1974)'s heuristic solution to distributed network control. To avoid computing Bellman's equation at many grid points that eventually poses the problem of dimensionality, the heuristic solution 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 visited. By evaluating all the subsets that can be possibly visited, PRODYN calculates the optimal trajectory of control policy in a planning horizon of 75s. The process then rolls forward one step in time. Experiments (Henry 1989) showed that PRODYN yields an average reduction in total travel time of 10%.

UPTOPIA (Urban Traffic OPtimisation by Integrated Automation, Mauro et al., 1989) is a hybrid control system that combines online dynamic optimisation and offline optimisation. This is achieved by constructing a system hierarchy that has an area level and a local level. The area controller generates a 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, and is 120s long, with the process being repeated every 3 seconds. 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 typical traffic flows for various parts of the day. Afterwards AUT prepares the data for TRANSYT calculation and starts TRANSYT optimisation. The benefits recorded after UTOPIA's implementation show an increase of 15% in average speed for private vehicles and 28% for public transport with priority.

The following two systems differ from the cases reviewed above. They are not based on state-space representation of system and do not involve the concept of DP. However, they do employ optimisers to calculate signal timings in respect to a set of performance measures.

MOVA (Vincent and Peirce, 1988) is purposely designed for dynamic operation at isolated intersection. The system generates signal timings cycle-by-cycle to optimise an objective function, which is to minimise delay and stop in an uncongested situation and maximise capacity in a congested situation. 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. MOVA updates its signal plans every half-second.

SCOOT (Hunt *et al.*, 1982) is a centralised adaptive system developed in U.K. by the Transport Research Laboratory. The SCOOT system optimises green time splits, offsets, and cycle length separately. The split optimiser equalises saturation in an intersection by minimising the maximum degree of saturation on links approaching the intersection. The degree of saturation is calculated by using the traffic Flow Profiles. This optimiser runs 5 seconds before the current green stage expires, and decides whether a stage should start 4 seconds earlier, remain the same, or start 4 seconds later. Offset optimiser once per cycle uses the Flow Profiles to predict performance measures throughout a cycle for an intersection. These predictions are used for evaluating three options for offset: reduce offset by 4 seconds, keep the current offset, or increase by 4 seconds. The cycle length optimiser operates on a region of intersections that are expected to have good progression between them. It looks at the degree of saturation for all links in the region. If those degrees are at ideal level, the optimiser increases the Minimum Practical Cycle (MPCY) length for each intersection by a small fixed step, and if all degrees are below ideal level, the MPCY is reduced by a small fixed step.

Systems reviewed above respond to the changes in traffic with adjusted control variables. They are characterised by the feedback path of the control output, and constantly monitor the traffic. However, they do not engage machine learning in the feedback path, and do not adjust control polices or parameterised value functions accordingly. These systems do not evolve as information of control environment accumulates. In the next section, we review methods that explore machine learning in adaptive traffic signal control.

2.3 Developing Methods in Adaptive Traffic Signal Control

This part of review focus on methods engaging machine learning in adaptive traffic signal control. These control systems can be thought of as having two loops. One loop is a normal feedback with the process and the controller. The other loop is the parameter adjustment loop. A machine learning technique is usually required to supervise the parameter adjustment. There are broadly two trends in developing this sort of adaptive controllers, one focusing on using heuristic based AI techniques, the other on reinforcement learning.

2.3.1 Heuristic based methods

Papis and Mamdani (1977) used fuzzy logic method to develop a controller for isolated intersection. Applications of fuzzy control in traffic networks were studied by Nakatsuyama *et al.* (1984), Nakamiti and Gomide (1996), Nakamiti and Freitas (2002) and Srinivasan *et al.* (2006). Generic reinforcement learning techniques for network optimisation were studied by Mikami and Kakazu (1994). Spall and Chin (1997) used artificial neural network (ANN) to map traffic patterns to cyclic signal timings, and used perturbation algorithm to obtain reinforcement signal to adjust neural weights.

A common feature for the heuristic solutions above is that traffic signal controller learns the mapping of detected traffic patterns to signal timing plans. The plans are usually stored in a library, and are retrieved to meet the prevailing traffic patterns. The learning process can be either performed offline or online. The traffic patterns are usually preset, and the signal timings cyclic. The “black-box” effects are usually associated with fuzzy logic control and generic solutions. This makes the solutions case-sensitive, and difficult for generalisation.

2.3.2 Reinforcement learning related methods

RHODES (Real-time Hierarchical Optimized Distributed Effective System, Mirchandani and Head, 2001) is a hierarchical adaptive traffic signal control system that addresses dynamic network loading, network flow control and intersection control as three operations layers. At each level of the hierarchy there is an estimation/prediction component and a control component. The predictions are processed by a specific model that uses detected real-time information to estimate link free-flow speed, queue discharge rates, turning probabilities, and characteristics of platoons. Online adjustment to these estimates influences the control component in decision-making. The intersection controller uses dynamic programming (DP), and defines state variable i as the amount of time that has been allocated to all past phases 1, 2, …, j. The decision in phase j is to allocate u_j time units to the current phase. The DP algorithm is completed when each possible decision for each phase has been evaluated in a forward recursion. Then backward recursion is used to determine the sequence of phases and their durations. The network flow controller enumerates all possible decisions to accommodate conflicting demand from traffic platoons. The RHODES system reports 30-50% reduction in delay from semi-actuated systems in tests by simulation. The key issue with RHODES, however, is the computational burden on the intersection controller. The definitions of the state and decision variables register a dimension of u^{2j}, and the DP approach computes for each possible decision for each phase. If there are 10 optimal allocation of time units for each phase, and 4 phases in total, the intersection controller performs 10^8 computations before rolling forward to the next step, not to mention that the network controller has to enumerates all possible decisions for managing platoons. It is unclear from the literature on how the system handles the computation demand in real-time.

Teodorovic *et al*. (2006) used ANN to map future traffic arrival pattern to green time extension for the current phase. For each pattern of vehicular arrivals, DP is used to find green time splits for all the approaches that results in optimal performance. The complete pattern-to-split dataset is then used to train the neural network, which then generates the mapping from pattern to action. As the training is performed offline, the implementation of the heuristic based on neural network demands negligible CPU times for computation. This approach produces a track of decisions that are as good as DP. The shortcoming of this approach is that the neural network training is offline, and traffic patterns have to be identified manually. A preset state sequence is required to formulate both DP and neural network solutions.

Cai (2007) proposed an approximate dynamic programming (ADP) solution for an isolated intersection. This approach addresses issues of dimensionality and incomplete information that arise in applying DP in real-time. Rather than using a heuristic to replace the principles of DP, the ADP approach uses a linear function to approximate the value function. As a result, the dimension of the state-space is substantially reduced to the size of a few functional parameters. This approach adopts a forward rolling process that uses limited online formation (10s of future vehicle arrivals) to plan ahead and update approximation progressively. Perturbation learning is used to update the approximation, which perturbs the system with incremental changes in state variables. Experimental results showed that the performance of this preliminary ADP controller is as good as existing adaptive control methods.

Heydecker *et al.* (2007) extended the same ADP approach to more complex intersection layouts. The results showed that the ADP approach reduced about a half in vehicle delays by comparison with the best fixed-time plans.

Li *et al*. (2008) presented another approach to develop ADP solution to traffic signal control. They used action dependent heuristic dynamic programming (ADHDP) approach initially developed by Werbos (1992). This approach uses two neural networks, one for mapping state to action, and the other for mapping state to discounted future value. Despite the initiative, this preliminary study lacks clarity in the definition of state and objective function. The study does not include performance comparison with established methods.

Cai *et al.* (2009) generalised the ADP control algorithm for isolated intersections. They use ANN as a universal approximator to value function and reinforcement learning as the learning paradigm. This study provides a generic definition of traffic state, controller state, state transition dynamics, and approximation function. The ANN based approximation structure can approximate both linear and non-linear value functions. This is particularly useful for expanding investigation because many delay functions (Webster 1957; Kimber and Hollis, 1979) are non-linear. This general ADP algorithm allows various machine learning techniques to be employed. *Temporal-difference* (TD) learning and perturbation learning were studied as two examples. The two learning techniques produced equal performance in experiment. The general ADP algorithm is compatible with a range of system resolutions, from 0.5s to 5s per temporal increment. Numerical results showed that reduction in vehicle delays from the optimised fixed-time plans is 43% at 5s resolution, and 67% at 0.5s resolution. Fundamentals of this approach and the numerical experiments are presented in the rest part of this thesis.

2.4 Discussions

In this chapter we reviewed established traffic control systems as well as those that are being developed. This review suggests that traffic-responsive systems are more effective in managing traffic at intersections than fixed-time systems, and that optimised responsive systems are better for non-stationary traffic environment than non-optimised ones. The state-of-the-art in adaptive traffic signal control includes using state-space presentation and sequential decision making at real-time. The advantage of using state-space presentation is that it applies to both linear and non-linear systems, and convenient to model multiple-input, multiple-output process. Sequential decision-making is preferable in traffic signal control because it offers the capability of effectively adapting to the evolutionary traffic. For a system represented in state-space, dynamic programming (DP) is the ideal solution to find the optimal sequence of decisions successively in time. However, the DP solution challenges traffic engineers in its computational difficulty and demand of complete information. These challenges make DP inadequate for real-time control, and approximation is the second-best solution.

Early studies in approximating DP derived approximation function from regression analysis, and did not engage mechanisms to adjust approximation online. Recent studies began to use real-time machine-learning techniques to adjust parameters of the approximation structure. The family of reinforcement learning fits well with approximate dynamic programming (ADP) because they use the formulae of DP to calculate learning signals and propagate the learning signals back to adjust the approximation structure. Reinforcement learning does not require the provision of ideal *target* or output for learning. Instead it learns from the trial-and-error process.

The rolling-horizon approach has been widely used in optimised traffic-responsive systems. It is particularly useful in a process where information of future traffic emerges successively in time. Controller calculates signal plans into the future, but only implements the signal plans for the current time, and then rolls forward to repeat calculation. In discrete systems, the smaller the time step, the finer the *resolution* of the system. Finer resolution means faster revision of signal plans, hence quicker response to changes in traffic. The rolling-horizon approach can be integrated to the ADP algorithm. This simply implies that the control algorithm calculates timings for the planning horizon at every time step, and implements the calculated timings only for that step.

The next chapter presents the fundamentals of ADP and machine learning techniques.

CHAPTER 3 APPROXIMATE DYNAMIC PROGRAMMING

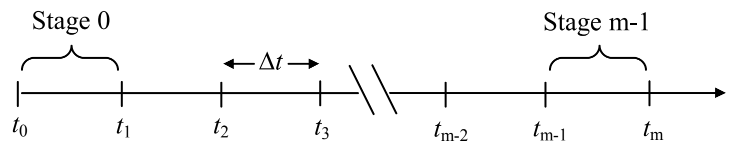
In this chapter, we investigate the fundamentals of the ADP concept, based on which we present a practical solution for real-time traffic signal control. We begin with definitions in Section 3.1 We then discuss the limitations of DP in practical use in Section 3.2. On the understanding of the DP formulae, we introduce the basic features of the ADP concept in Section 3.3. The completion of the ADP algorithm requires proper approximation architectures as well as appropriate real-time machine learning techniques to update the components of the architecture. These two important building blocks of ADP are discussed in Section 3.4. To show that the ADP concept is not limited to single architecture of approximation, we discuss an alternative method that explores structural properties of the original DP problem in Section 3.5. A summary of this chapter is provided in Section 3.6.

3.1 Definitions

We define the following variables and parameters for the discussion on dynamic programming:

i	is a vector of system state,
$J(i)$	is the exact value function (cost-to-completion) associated with state i,
u	is a decision vector,
U	is the decision space,
$J(i, u)$	is the exact value function associated with state i and decision u,
w	is a column vector of traffic arrival information,
W	is a vector of time-dependent traffic arrival information,
$p(j\|i,u)$	is the transition probability from state i to j by implementing decision u,
P	is a matrix of transition probabilities,
α	is a discount factor,
$g(\cdot)$	is a one-step cost function,
Δt	is a discrete time interval, and $t_m = t_{m-1} + \Delta t$.

For a control problem defined on time series, dynamic programming decomposes the problem into stages, each corresponds to successive discrete time epoch on the time serie. The time serie and stage defination can be shown as:



In each stage, dynamic programming evaluates decisions $u_t \in U$, and impletes optimal decision u^*_t. With exogenous information process $\{ w_0 , w_1 , \dots , w_t \}$ and decision u^*_t, the system is transferred from state i_t to i_{t+1}. This control process can be expressed as:

$$\mathfrak{F} = \{ i_0, u_0^*, w_0, i_1, u_1^*, w_1, \dots, i_{m-1}, u_{m-1}^*, w_{m-1}, i_m \} .$$

Dynamic programming is the only exact solution for the control process described above. However, its application for many real-time control problem is restricted. We extend discussion on this with details in the following section.

3.2 Limitations of Dynamic Programming

Given the initial state i_0 and a sequence of decisions u_t at discrete time t, a dynamic programming algorithm is to solve

$$\min_{u_t \in U} \underset{W}{E} \left\{ \sum_{t=0}^{m-1} \alpha^t g(i_t, i_{t+1}) | i_0 = i \right\}. \tag{3-1}$$

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

$$J(i_t) = \min_{u_t = U} \underset{w_t}{E} \{ g(i_t, i_{t+1}) + \alpha J(i_{t+1} | i_t) \}, \text{ for } t = m-1, m-2, \dots, 0, \tag{3-2}$$

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} with decision u_t. J (i_t) values are stored in a look-up table, where the dimension of the table is equal to the dimension of the state space. Each cell of the multi-dimensional table corresponds to a cost-to-completion value from a certain stage.

To allow this approach to be solved coherently, its problem *should* have the *Markovian property*. A process is said to have the Markovian property if

$$P\,\{\, i_{t+1} = j \mid i_0, i_1, \ldots, i_t \,\} = P\{\, i_{t+1} = j \mid i_t \,\},$$

for $t = 0, 1, \ldots$ and every sequence $i, j, i_0, i_1, \ldots, i_{t-1}$. This implies that knowledge of the current state of the system conveys all the information about its previous behaviour necessary for determining the optimal policy henceforth. In case where the Markovian property holds, backward dynamic programming guarantees that an optimal policy for the whole problem is found when state i_0 is reached.

Despite the simple form exhibited by the Bellman equation and the global optimality it guarantees, dynamic programming is often of little practical value. A problem formulated in dynamic programming usually cannot be solved analytically, and computational requirement for finding optimal solution numerically is in exponential order to the size of state space. Additionally, a complete set of information for the whole problem is required. For operations at real-time, we usually do not have complete information a priori.

To show the problem of dimensionality associated with DP, let us consider a problem that has state variable i_t, information variable w_t, and decision variable u_t.

1. The state space. If state variable $i_t = (\, i_t(1),\ i_t(2),\ \ldots,\ i_t(K)\,)$ has K dimensions, and supposing that each $i_t(n)$ takes one of M_i possible values, the total number of states at each step t is M_i^K.
2. The information space (or the space of random noise). If the information variable $w_t = (\, w_t(1),\ w_t(2),\ \ldots,\ w_t(L)\,)$ is L-dimensional, and each $w_t(n)$ takes one of M_w possible values, the size of information space is M_w^L.
3. The decision space. If decision variable $\underline{u_t} = (\, u_t(0),\ u_t(1),\ \ldots,\ u_t(N)\,)$ has N dimensions, and each $u_t(n)$ may take M_u possible values, the total number of eligible decision is M_u^N.

The Bellman equation requires that at each step t, for each $i_t(n)$, $w_t(n)$ and $u_t(n)$, a J value is calculated so that an optimal decision can be made. The computational requirement for this numerical solution is

$$M_i^K \times M_w^L \times M_u^N . \tag{3-3}$$

In the case that K=10, L=5, and N=5, and M_i^K= M_w^L= M_u^N=10, the total amount is

$$10^{10} \times 10^5 \times 10^5 = 10^{20} . \tag{3-4}$$

Powell (2007) referred to this as the ‘three curses of dimensionality.’ Such a computational requirement makes dynamic programming impractical for real-time operation, because the time it takes to finish an iteration may well exceed the duration of discrete time increment Δt. This is a direct concern for controlling traffic signals at real-time. The time window for evaluating and implementing a decision is generally not more than 5 seconds.

The example presented by Eq. (3-4) is nowhere near a complicated problem. A problem with larger state space may well become computationally intractable in dynamic programming. In the case of traffic signal control, Henry *et al.* (1983) found that, for a traffic intersection of 4-link only, the memory requirement for a look-up table approximation of J values is “tremendously high” and impossible for a controller to find optimal solutions at real-time.

Another major obstacle in applying DP for real-time control is incomplete information. In real-time optimisation, the incomplete information may refer to knowledge of underlying model of the control process, the state transition probability, or the exogenous process used as system input. In our case, we concern about the traffic arriving information, which is exogenous to the control process and used as input to the system. The recursive calculation using Eq. (3-2) assumes that *complete* information is available up to time m so that the calculation can start from step $m - 1$. The complete information includes exact knowledge of exogenous process and distributions of state transition model. In reality, real-time information about arriving traffic is obtained from sensors in short advance, normally not more than 10 seconds in urban area. Although using DP for the 10-second optimisation problem is a possible solution, this prevents the controller from interacting with the control process to accumulate knowledge over time, thus reducing the advantage of adaptive control.

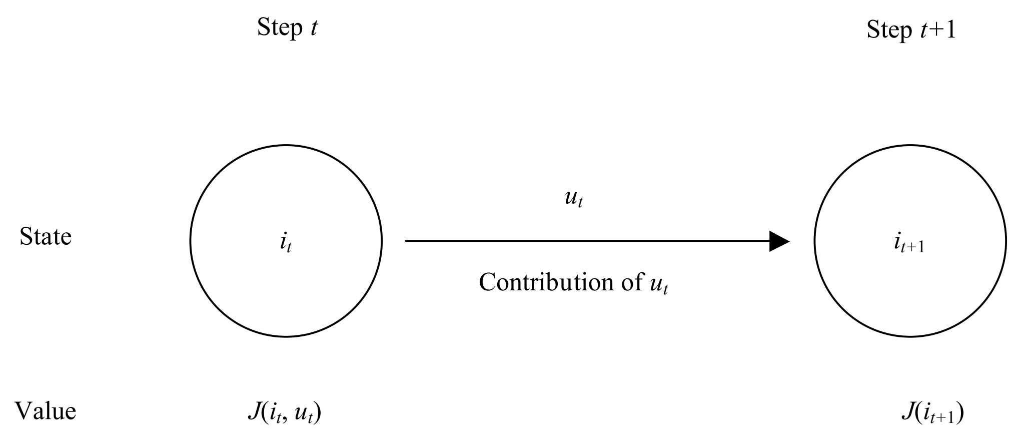


Fig. 3-1 A sample state transition of deterministic dynamic programming. State is transferred from i_t to i_{t+1} by implementing decision u_t.

In order to bring the principles of DP to real-time control, there are two immediate tasks: first, reducing the dimensionality of the control problem; second, accumulating limited sensor information progressively to improve knowledge of underlying control process. In the next section, we show the concept of ADP addresses the tasks.

3.3 Fundamentals of Approximate Dynamic Programming

Approximate dynamic programming is a derivative from DP. The fundamentals of ADP, therefore, are derived from those of DP. In this section, we expand the discussion on the fundamentals of DP to show ADP as a general approach to solve the problems of DP in real-time control, rather than being a specific solution. Section 3.3.1 contains discussions on deterministic and stochastic dynamic programming. Section 3.3.2 introduces DP formulae for finite and infinite problems. Some shorthand notations are introduced in Section 3.3.3 to facilitate further discussions. Section 3.3.4 introduces the iteration algorithms to solve DP with infinite horizon. The basic features of ADP are introduced in Section 3.3.5.

3.3.1 Deterministic and stochastic DP

According to the lauguage of dynamic programming, a deterministic problem is one in which the state at the next step is completely determined by the state and policy decision at the current step. The Bellman equation for a deterministic problem can be written as

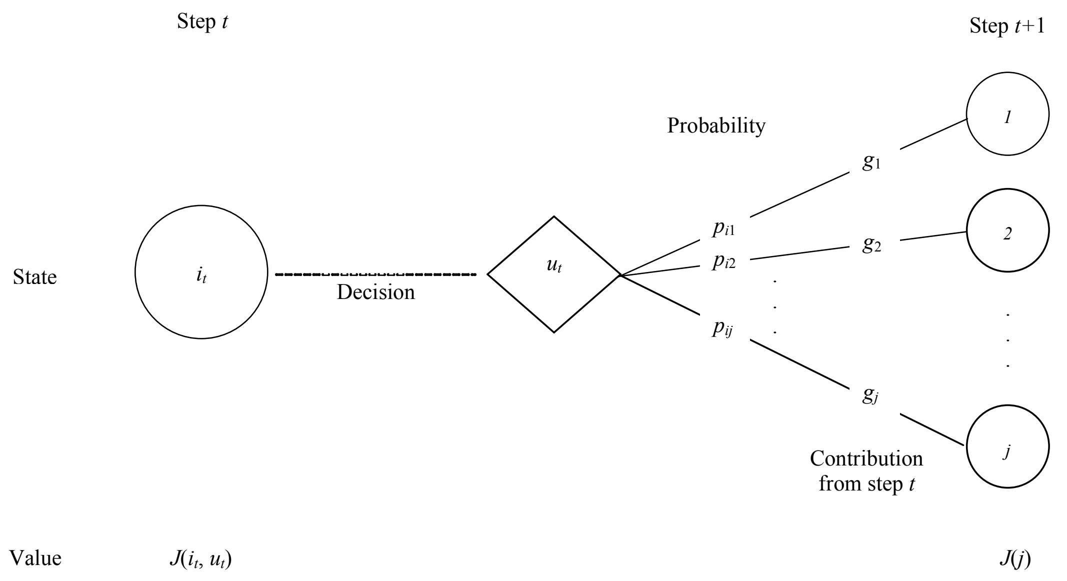


Fig. 3-2 A sample of state transition in probabilistic dynamic programming. The transfer from state i_t by implementing decision u_t follows a probabilistic distribution. Value $J(i_t)$ is the expected value of making decision u_t at state i_t.

$$J(i_t) = \min_{u_t \in U} \left\{ g(i_t, i_{t+1}) + \alpha J(i_{t+1} | i_t) \right\} . \tag{3-5}$$

Comparing with Eq. (3-2), the difference here is that there is no need to include the expectatoin operator E. Deterministic dynamic programming is shown diagrammatically in Fig. 3-1.

A stochastic (or probabilistic) problem differs from a deterministic one in that the state at future steps is not completely determined by the state and policy decision at the current step. The transition in states follows a *probabilistic distribution*. However, this transition is consistent with Markovian property, because the probabilistic disstribution can still be completely determined by the current state and the policy decision made at that step. Assuming that state variable i is K-dimentional, the Bellman equation can be reorganised for probabilistic problem as

$$J(i_t) = \min_{u_t \in U_t} \sum_{i_{t+1}=1}^{K} p_t\left(i_{t+1} | i_t, u_t\right)\left(g(i_t, i_{t+1}) + \alpha J(i_{t+1})\right) \text{ for } i_t = 1, 2, ..., K . \tag{3-6}$$

Fig.3.2 shows diagrammatically the probabilistic state transition.

As for traffic signal control, we have the following principal assumptions:

Assumption 3-1 *At each time step t, the system receives some sensory information of future vehicle arrivals w_t before a decision u_t is required.*

Assumption 3-2 *Vehicle arrivals form a Poisson process, which is exogenous to the traffic signal control system.*

Assumption 3-1 reflects the norm in real-time traffic signal control, where roadside sensors can be installed upstream of the intersection to inform controller of future arrivals. The controller may accordingly make adjustment to signals before the detected traffic have arrived at the intersection. Under this assumption, the state at the next time step is determined by the system state x_t, information w_t and policy decision u_t at the current step. The state transition at each step is deterministic.

However, Assumption 3-2 gives rise to a stochastic process for arriving. Since state transition is influenced by random arriving traffic, the process of $\{i_t\}$ can be seen as a stochastic process with Markov property, i.e. a Markov process.

3.3.2 Finite and infinite horizons

A problem formulated in dynamic programming is said to have a *finite horizon* if the value function $J(\cdot)$ accumulates over a finite number of steps, say m, which can be expressed as

$$J(i_0) = \min_u E\left\{\alpha^m h(i_m) + \sum_{t=0}^{m-1} \alpha^t g(i_t, i_{t+1}) | i_0\right\}, \tag{3-7}$$

where $h(i_m)$ is a terminal cost for ending at final state i_m . A finite problem formulated in dynamic programming can be solved numerically through steps $t = m$-1, m-2, …, 0 by recursively calculating Eq. (3-2). Problems of this sort often bear interest of achieving optimisation over a specific horizon. A good example of such is the shortest path problem.

Similarly, a problem is said to have an infinite horizon if value function J $(\cdot)$ accumulates infinitely, which can be expressed as

$$J(i) = \min_{u_u \in U} E\left\{\sum_{t=0}^{\infty} \alpha^t g(i_t, i_{t+1}) | i_0 = i\right\}. \tag{3-8}$$

The infinite horizon problem is of particular interest to understand steady-state properties in Markov process. At steady-state, the state transition probabilities become time-invariant, and the value of $J(\cdot)$ converges. The steady-state properties are important to derive analytical solutions to the control problem. Solving the infinite horizon problem, requires an iteration algorithm that leads to convergence in values of $J(\cdot)$ and a stopping criterion that specifies the region of convergence. There are two common iteration algorithms: *value iteration* and *policy iteration*. The two algorithms are discussed in Section 3.3.4.

Since the solution to DP formulae of finite horizon is arrived differently from that of infinite horizon, it is important to distinguish the properties of horizon for a control problem. As for traffic signal control, if we take the problem as of finite horizon, difficulty arises from establishing the terminal cost $h(i_m)$, which presents the long-term influence of current decisions. There is no existing approach to determine terminal cost analytically in the studies of vehicle queuing at signalised traffic intersection. Alternatively, if the control problem is formulated with infinite horizon, we may use iteration algorithms to accumulate knowledge over time and eventually achieve convergence to exact values of $J(\cdot)$. In this regard, we formulate traffic signal control with infinite horizon in this study.

3.3.3 Shorthand notations

Before proceeding with the discussion on iteration algorithms that solve a dynamic programming problem of infinite horizon, it is convenient to introduce some shorthand notations in expressions what would be otherwise complicated to write.

We define an operator T, $\forall\ i \in X$, that

$$(TJ)(i) = \min_{u_t \in U_t} \sum_{i_{t+1}=0}^{N} p_t\left(i_{t+1} \middle| i_t, u_t\right)\left\{g\left(i_t, i_{t+1}\right) + \alpha J\left(i_{t+1}\right)\right\}, \tag{3-9}$$

where TJ produces a vector, and $TJ\,(i)$ refers to element i of this vector.

Let us further define a $k \times k$ transition matrix P whose ij^{th} entry is $p_{ij}(\,j \mid i,\, u)$. In vector form, we can write TJ as

$$TJ = \min P\{g + \alpha J\}. \tag{3-10}$$

Let *S* be the space that contains all value functions, then, operator *T* is a mapping

$$T : S \to S . \tag{3-11}$$

An iteration algorithm that maps *J* to *TJ* for *m* iterations can be then denoted as

$$\left(T^m J\right)(i) = \left(T\left(T^{m-1} J\right)\right)(i), \forall i \in X . \tag{3-12}$$

where for convenience we write

$$\left(T^0 J\right)(i) = J_0(i),\ \forall i \in X . \tag{3-13}$$

For a specific policy μ , we define an operator T_μ such that

$$T_\mu J = P_\mu\left(g + \alpha J\right), \tag{3-14}$$

for vector $J \in S$.

This shorthand notation is useful for showing mathematical proofs, and particularly for the proof of convergence of iterations algorithms. However, this notation is not used for describing models and control algorithms related to traffic signal control in this study.

3.3.4 Iteration algorithms

There are two common iteration algorithms for solving infinite DP problems. One is *value iteration* and the other is *policy iteration.*

Value iteration is a process that generates a sequence of $T^n J$ starting from an initial vector J_0, which is expressed in Eq. (3-12) and (3-13). Value iteration algorithm is summarised in Fig. 3.3. Because that value iteration requires an infinite number of iterations to obtain the exact vector *J*, a termination criterion is required, which in Fig. 3.3 is express as

$$\left\| T^m J - T^{m-1} J \right\| < \xi\left(1-\alpha\right)/2\alpha , \tag{3-15}$$

where ξ is a specified error tolerance, α is a discount factor, and $\|TJ\|$ is the max-norm defined by:

$$\left\| TJ \right\| = \max_{i \in X} \left| \left(TJ\right)(i) \right| , \tag{3-16}$$

```
Step 1.   Initialisation
          Set $J_0(i) = 0,\ \forall i \in X$.
          Choose an error tolerance parameter $\xi > 0$, and a discount factor $\alpha \in (0,1)$.
          Set $m = 1$.
Step 2.   For each $i \in X$ compute
              $\left(T^m J\right)(i) = \left(T\left(T^{m-1}J\right)\right)(i)$.
Step 3.   If $\left\| T^m J - T^{m-1} J \right\| < \xi(1-\alpha)/2\alpha$, $J = T^{m-1}J$, process stops;
          else $m = m + 1$ and go to step 2.
```

Fig. 3-3 The value iteration algorithm for dynamic programming problem of infinite horizon; shorthand notation T is used.

Therefore, $\|TJ\|$ is the largest absolute value of a vector of elements. The value iteration process is terminated if condition in Eq.(3-15) is satisfied. The proof of convergence of value iteration can be found in Powell (Section 3.9.2, 2007).

Policy iteration is an alternative to value iteration, and always terminates finitely. This iteration algorithm is popular in cases where value of a specific policy is to be found. Policy iteration starts from a initial policy μ_0 and generates a sequence of new policies μ_1, μ_2, …, μ_m. A policy iteration has two steps, the first is the *policy evaluation* step that calculates

$$J_{\mu_m} = T_{\mu_m} J\ , \tag{3-17}$$

and the second is the *policy improvement* that performs

$$\mu_{m+1} = \arg TJ_{\mu_m}\ . \tag{3-18}$$

This process is repeated with μ_{m+1} used in place of μ_m until we have

$$J_{\mu_{m+1}}(i) = J_{\mu_m}(i), \forall i$$

A stopping criterion such as stated by Eq.(3-15) may also apply here to increase the speed to convergence. The proof of the convergence of policy iteration can be found in Puterman (Theorem 6.4.6, page 180, 1994) and Bertsekas and Tsitsiklis (Section 2.2.3, 1995).

Both of the two common iterative algorithms are widely adopted in ADP techniques. A rich field in ADP called *functional approximation* (or J function approximation) is primarily based on value iteration. On the other hand, look-up table approximation techniques that establish state-actions pairs are usually built on policy iteration.

3.3.5 Features of ADP

Discussions in previous sections introduced the fundamentals of the original dynamic problem that is to be approximated by ADP. There are three basic features of ADP, regardless of the specific applications.

First, ADP aims to significantly reduce computational requirement so that the original DP problem becomes tractable in solution and feasible in implementation.

Second, ADP adopts a forward process rather than the backward recursive calculation. This forward process, i.e. stepping forward in time, allows ADP to use information becomes available between time *t* and *t*+1 to facilitate decision-making. It is normally assumed in ADP research that the observation of exogenous information, such vehicle arrivals, becomes available after a decision is made, but before the next decision epoch. To make an optimal decision without detailed information can be facilitated by *Monte Carlo* simulation, which generates a sample path for the system. A decision calculated for the sample path takes the system to a *post-decision state*, which is referred to be the state immediately after the decision is implemented. State transition only happens after specific information is observed.

Third, the approximation that is used in ADP improves progressively. It is common that the exact values of functional parameters are not known a priori. Upon each observation of state transition, the ADP algorithm updates the parameters of the approximation function by applying certain learning techniques.

In this thesis, a continuous function is employed to approximate the *J* values, thus replacing a look-up table of *J*(*i*). This substantially reduces computational requirements, and makes ADP practical for real-time implementation. The system process is stochastic because it is influenced by exogenous and random vehicle arrivals. The state transition is deterministic because the system receives real-time information of future arrivals before evaluating a decision, as stated in Assumption 3-1. This feature avoids the need to distinguish between pre-decision and post-decision states. To make this practical, we assume no errors in data detection and transmission in the system.

We define a continuous approximation function $\tilde{J}(\cdot, r): X \times \mathbb{R}^K \rightarrow \mathbb{R}$ to replace the exact $J(\cdot): X \rightarrow \mathbb{R}$, where *r* is a *K*-dimensional parameter vector of $\tilde{J}$, and *X* the state space. By indexing successive states with positive integers, we can view the state space as the set *X* = {1, …, *n*}, where *n* is possibly infinite. The sequence of states visited by the stochastic process is denoted by $\{i_t \mid t = 0, 1, \ldots\}$. At each discrete temporal interval *t*, we have a new observation of state transition and calculate

$$u_t^*(i_t) = \arg\min_{u_t \in U} E\left\{ g(i_t, i_{t+1}) + \alpha \tilde{J}(i_{t+1}, r_t) \right\}, \tag{3-19}$$

and record

$$\hat{J}(i_t) = \min_{u_t \in U} E\left\{ g(i_t, i_{t+1}) + \alpha \tilde{J}(i_{t+1}, r_t) \right\}. \tag{3-20}$$

After a state transition is observed, a common objective in updating approximation function is to find

$$r^* = \arg\min_{r \in \mathbb{R}^K} \left\| J - \tilde{J} \right\|, \tag{3-21}$$

by incrementally calculating correction signal Δ*r* and updating estimation through

$$r_{t+1} = r_t + \eta_t \Delta r_t, \tag{3-22}$$

The correction signal Δr_t is usually obtained from machine learning process.

Convergence of *r* to r^* by using an incremental process described by Eq. (3-22) is guaranteed if specific approximation structures and learning techniques are used. Theories and assumptions for the convergence are further discussed in the next section. A general ADP algorithm is illustrated in Fig. 3-4.

Using approximation function $\tilde{J}(\cdot, r)$, we avoid the need of a look-up table of $J(i)$. This reduces computational requirement from a magnitude exponential in state space size to a magnitude that is polynomial in state space size. It is easy to calculate the reduction in computation through Eq. (3-3) and (3-4). Other variables being equal, let us assume both state *i* and parameter *r* of the function $\tilde{J}(i, r)$ are of dimension 10, the computational requirement for making

```
Step 1.   Initialisation
          a) Initialise $r_0$.
          b) Choose an initial state $i_0$, $i \in X$.
          c) Set $t = 1$.
Step 2.   System receives random noise $w_t$.
Step 3.   For $t = 1,2,É\ , m-1$,
          a) Calculate
                 $\hat{J}(i_t) = (T\tilde{J})(i_t, r_t)$
          b) Calculate
                 $\Delta r(\hat{J}(i_t))$
          c) Update parametric vector $r$
                 $r_{t+1} = r_t + \eta_t \Delta r(\hat{J}(i_t))$
          d) transfer system to new state $i_{t+1}$.
Step 4.   If $t < m$ go to step 2.
```

Fig. 3-4 A generalised approximate dynamic programming algorithm

$$10 \times 10^5 \times 10^5 = 10^{11}, \tag{3-23}$$

which is 10^9th of the requirement in classic dynamic programming described in Eq. (3-4).

In comparison with other contemporary adaptive traffic signal control methods, the features of ADP offer the following advantages. First, using state-space representation, the ADP method establishes a higher degree of awareness of the prevailing traffic condition. Second, it achieves sequential decision-making at real-time, which gives controller larger freedom to respond to stochastic vehicle arrivals. The frequency of decision-making can be improved by using a higher resolution for the discrete time system. Third, the mechanism for updating approximation function provides better prediction of the impact of decision from current state, which serves as a leverage to balance immediate control performance and that of the long-term. These advantages are likely to deliver better performance than other rule-based adaptive controllers in real-time operation.

3.3.6 Summary

In Section 3.3 we discussed the formulae of the original dynamic programming in a number of generic terms. These terms include the deterministic and stochastic problems, the finite and infinite horizons, and the value and policy iterations. Based this, we showed that the basic feature of the ADP concept is to replace exact J values with an approximation function

$\tilde{J}(\cdot,r)$ so that computational requirement can be reduced substantially, and reliance on complete information is relaxed. Starting with any arbitrary values of functional parameters *r*, we showed the incremental process of updating the parameters at real-time. The incremental process requires corrections to estimation, which are usually supplied by learning techniques. In the next section, we discuss approximation structures together with real-time learning techniques.

3.4 Learning to Approximate

An ADP strategy that uses $\tilde{J}(\cdot,r)$ in place of exact *J* values offers an open framework for theoretical development. Just as that there is no standard mathematical formulae for dynamic programming, there is no standard formulae for $\tilde{J}(\cdot,r)$. We will therefore investigate into appropriate architectures for approximation function. Once appropriate approximation architecture is established, learning techniques can then be applied to update parameters of the approximation function progressively. Approximation architectures are introduced in Section 3.4.1. General machine learning theories are introduced in Section 3.4.2, and specific ones that are used for this study in Section 3.4.3 and Section 3.4.4. Integration of learning techniques to the ADP value iteration algorithm is discussed in Section 3.4.5. A concise conclusion is provided in Section 3.4.6.

3.4.1 Approximators

In this thesis an *approximator* is a *continuous* approximation function to exact cost-to-completion values *J*. The group of approximators can be broadly classified into two categories: linear approximators and non-linear approximators. However, it is worth noting that continuous function is not the only way for *J* function approximation. A common form that is not continuous is *aggregation*, which belongs to categorical approximation.

Our interest in this study primarily lies in linear function approximation. There are two reasons: the first is that linear function is simple for both implementation and training; the second is that literatures on ADP with linear approximator are relatively rich and well proved. A separable linear approximator can be expressed as

$$\tilde{J}(i,r)=\sum_{j=1}^{K} r(j)\phi_j(i), \forall i \in X\,, \tag{3-24}$$

where $r = (r(1),\ r(2), \ldots, r(K))$ is a column vector with each entry a parameter of approximation function, and each ϕ_j is a mapping function defined on the state space X. The function ϕ_j can be viewed as feature-extraction function (or basis functions) that maps state to real-valued feature vector, while each $r(j)$ can be viewed as the associated weight. Feature function based approximation architecture is illustrated in Fig. 3-5.

Let $\phi'(i) = (\phi_1(i), \phi_2(i), \ldots, \phi_K(i))$, where the prime denotes transposition, the approximation can also be written as

$$\tilde{J}(i,r)=r'\phi(i),\ \forall i=1,\ldots,n\,, \tag{3-25}$$

or

$$\tilde{J}(r)=\Phi r\,, \tag{3-26}$$

where Φ is a $|X|\times K$ matrix whose jth column is equal to ϕ_j and,

$$\Phi=\begin{bmatrix} | & & | \\ \phi_1 & \ldots & \phi_K \\ | & & | \end{bmatrix}=\begin{bmatrix} - & \phi'(1) & - \\ & & \\ - & \phi'(n) & - \end{bmatrix}. \tag{3-27}$$

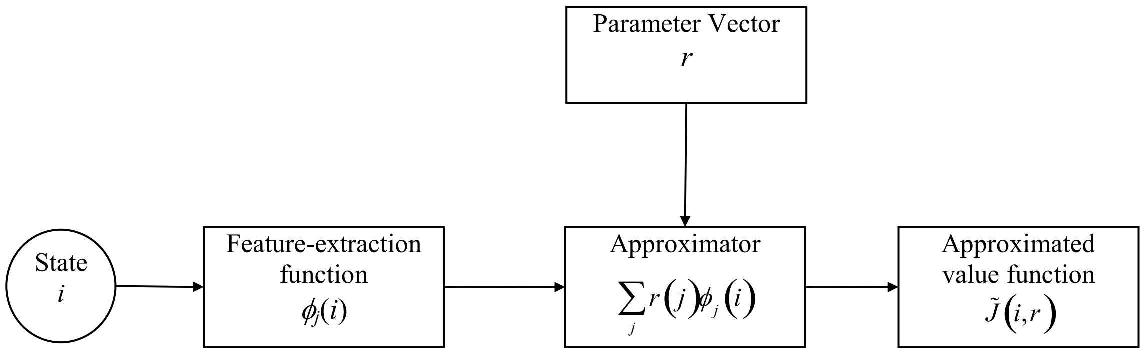

Parameter Vector r
State i
Feature-extraction function $\phi_j(i)$
Approximator $\sum_j r(j)\phi_j(i)$
Approximated value function $\tilde{J}(i,r)$

Fig. 3-5 A feature-extraction based approximation architecture

The first derivative with respect to the parameter vector r is given as

$$\nabla\tilde{J}(i,r) = \phi(i) \ , \tag{3-28}$$

and we have

$$\nabla\tilde{J}(r) = \Phi \ , \tag{3-29}$$

where $\nabla\tilde{J}(r)$ is the Jacobian matrix whose i^{th} row is equal to $\nabla\tilde{J}(i,r)$.

Non-linear approximation is a field of rich literature too. Here we limit the discussion to non-linear approximator based on ANN. According to Haykin (1999), an ANN is

A directed graph consisting of nodes with interconnecting synaptic and activation links, and is characterised by four properties:

1. *Each neuron is represented by a set of linear synaptic links, an externally applied bias, and a possibly non-linear activation link. The bias is represented by a synaptic link connected to an input fixed at* +1.
2. *The synaptic links of a neuron weight their respective inputs.*
3. *The weighted sum of the inputs defines the induced local field of the neuron in question.*
4. *The activation link squashes the induced local field of the neuron to produce an output.*

In mathematical terms, we can describe a neuron j by writing the following equations:

$$v_j = \sum_{i=1}^{k} r_{ji}x_i + b_j \, , \tag{3-30}$$

and

$$y_j = \varphi(v_j) \, , \tag{3-31}$$

where

$x_1, x_2, \ldots, x_k$ are inputs to neuron j;

$r_{j1}, r_{j2}, \ldots, r_{jk}$ here represent the synaptic weights of neuron j;

b_j is the bias;

v_j is the *induced local field;*

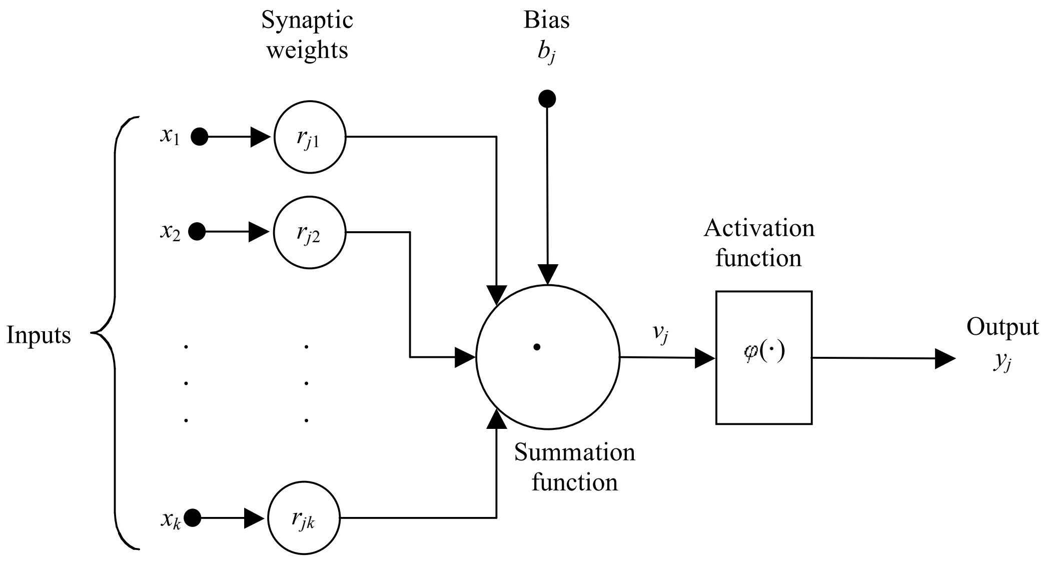


Fig. 3-6 A model of a neuron

$\varphi(\cdot)$ is the *activation function*;

y_j is the output of the neuron.

A model of neuron is shown graphically in Fig. 3-6. Depending on the activation function $\varphi(\cdot)$, a neuron can be either linear or non-linear. Without touching the profundity of the non-linear family, we present two typical examples: *threshold function* and *sigmoid function*.

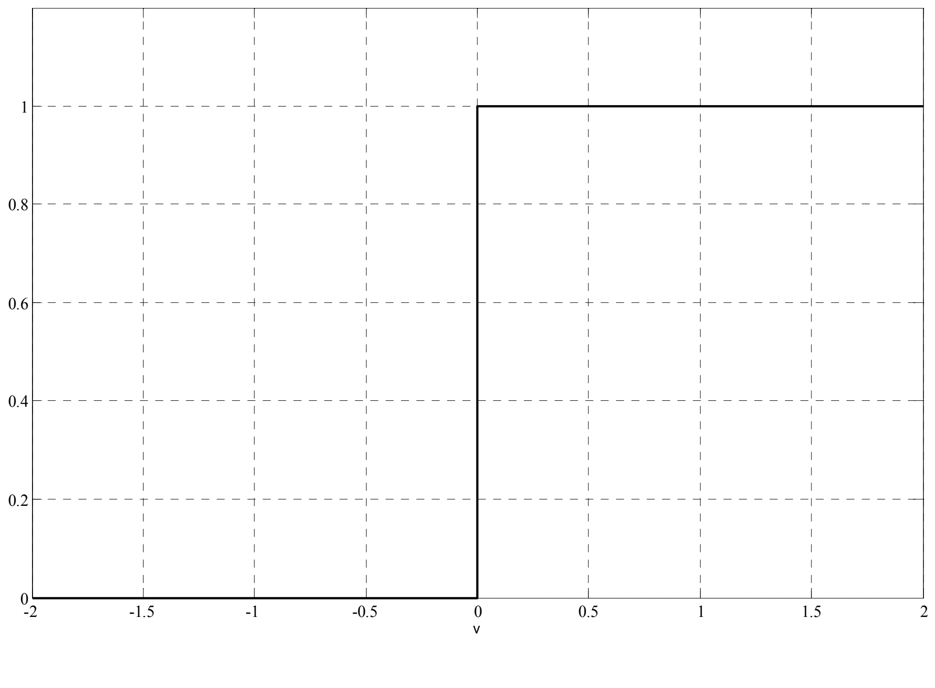


Fig. 3-7 Threshold function

Threshold function, which is described in Fig. 3-7, can be specified as

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}.$$

The sigmoid function is a common form of activation function. It is defined as a strictly increasing smooth function. An example of the sigmoid function is the *logistic function* (Fig. 3-8) defined by

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

where a is the slope parameter of the sigmoid function.

Neural networks are constructed on the inter-connected neurons. Typical architectures of neural networks include *single-layer feedforward* networks (Fig.3-9) and *multilayer feedforward* networks (Fig. 3-10). In feedforward networks, inputs propagate strictly from input layer to output layer, on a layer-by-layer basis. The pathways of information in neural networks are indicated by the arrows in Fig. 3-9 and Fig. 3-10. A layered network can be expressed mathematically as

$$y = \varphi\left(\sum_n r_{kn}\varphi\left(\sum_m r_{lm}\varphi\left(\ldots\varphi\left(\sum_i r_{ji}x_i + b_j\right)\right)\right)\right), \tag{3-32}$$

which maps $x \in \mathbb{R}^{n_X}$ to $y \in \mathbb{R}^{n_Y}$.

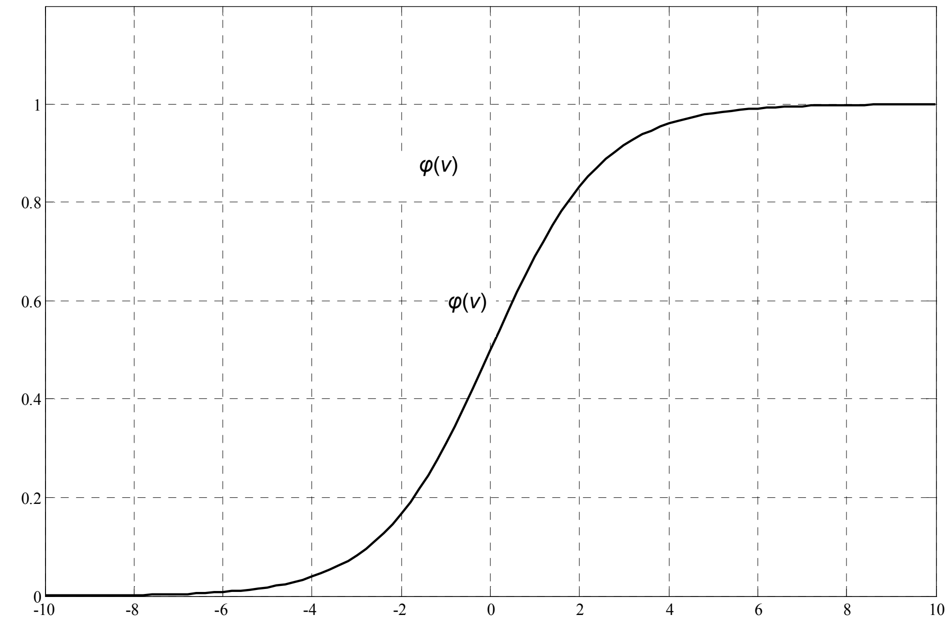


Fig. 3-8 Logistic function at a = 0.8

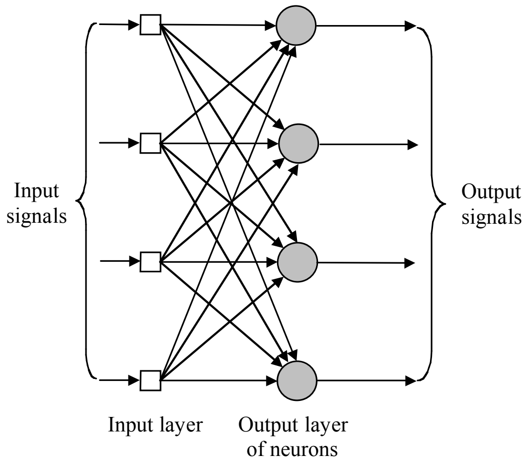


Fig. 3-9 A feedforward single-layer neural network

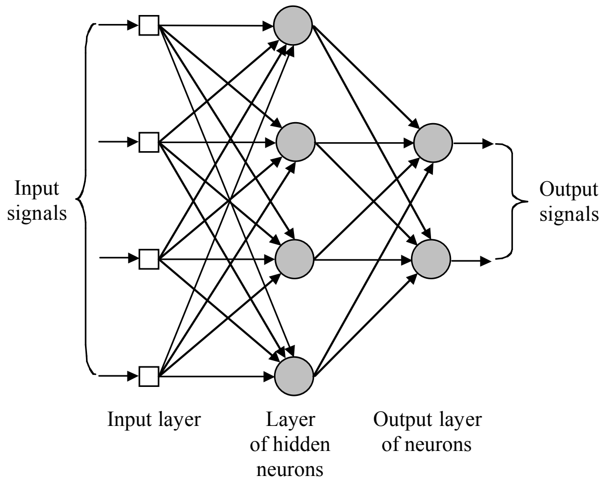


Fig. 3-10 A feedforward multilayer neural network

The scheme of nested sigmoid functions described in Eq. (3-32) is a *universal approximator* defined by *universal approximation theorem*, which can be stated as

Theorem 3.1. *Let $\varphi(\cdot)$ be a non-constant, bounded, and monotone-increasing continuous function. Let I_m denote the m-dimensional unit hypercube $\left[0,1\right]^m$. The space of continuous functions on I_m is denoted as $C(I_m)$. Then, given any function $F \in C(I_m)$ and $\varepsilon > 0$, there exists an integer n and sets of constants β_j, b_j, and r_{ji}, where $j = 1, \ldots, n$ and $i = 1, \ldots, m$ and function given by*:

$$f\left(x_1,\ldots,x_m\right) \triangleq \sum_{j=1}^{n} \beta_j \varphi\left(\sum_{i=1}^{m} r_{ji} x_i + b_j \right),$$

such that:

$$\left|F\left(x_1,\ldots,x_m\right)-f\left(x_1,\ldots,x_m\right)\right|<\varepsilon$$

for all x_1, x_2, …, x_m *that lie in the input space* I_m.

The proof of Theorem 3.1 is provided in Cybenko (1989). The universal approximation theorem is an *existence theorem* in the sense that it provides the mathematical justification for the approximation of an arbitrary continuous function as opposed to exact representation.

It is worth noticing that when a neural network consists of a single neuron *j*, Eq. (3-32) becomes

$$y=\varphi\left(\sum_{j=1}^{m}r_jx_j+b\right). \tag{3-33}$$

If φ is linear then linear relationship establishes between *x* and *y*.

3.4.2 Learning paradigms

In common practice, neural network weights are initialised arbitrarily. Learning techniques are required to update the weights.

The term *learning* in the context of neural networks is defined by Mendel and McClaren (1970) as:

> *"Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place".*

The paradigms of learning can be broadly classified into two categories: *supervised learning* and *unsupervised learning*. The former assumes that the exact output of a system is known externally, but unknown to the neural network, which is a part of the operating system. Each time the exact output is used to correct the output of the neural network, and an error term is generated. The error term is then propagated backward into the neural network to adjust neural network parameters, from output neuron node to the input layers, layer-by-layer. When the exact output is transferred to neural network through a number of iterations, we may then dispense with the external source and let the neural network operate in a self-sufficient manner.

The paradigm of unsupervised learning, as the name implies, pertains to the condition of having no exact output for the learning process to match. Unsupervised learning is aimed at a *task-independent measure* of the quality of representation, and the free parameters of the network are optimised with respect to that measure. Once the network has become tuned to the statistical regularities of the input data, it develops the ability to form internal representations for encoding features of the input and thereby to create new classes automatically. Central to unsupervised learning is *reinforcement learning*. Barto *et al.* (1983) defines reinforcement learning as:

> *"A 'behavioural' learning problem performed through interaction between the learning system and its environment, in which the system seeks to achieve a specific goal despite the presence of uncertainties".*

Trial and error search are important characteristics of reinforcement learning. According to Sutton and Barto (1998), a reinforcement learning system generally consists of four basic components: *a policy, a reward function, a value function,* and *a model of the environment.* The *policy* is the ultimate determinant of behaviours and performance. The *reward function* returns the immediate and defining feature of the problem faced by the agent. *Value function* predicts the rewards in the long run. And the *model of the environment* predicts (in stochastic environment) or determines (in deterministic environment) the next state. It can be seen that reinforcement learning is directly related to dynamic programming, as we may match the components of reinforcement learning to Eq. (3-1) and (3-2). Dynamic programming, therefore, provides the mathematical formalism for sequential decision making, whilst reinforcement learning provides the capacity for adjusting free parameters through the interaction with the environment. This combination of mathematical formalism and learning paradigms completes the theoretical framework of ADP. Without the knowledge of exact J values (otherwise we would just use dynamic programming to solve the problem), it is reinforcement learning that enables the ADP method to improve approximation through its interaction with the operation enviroment.

3.4.3 Neural network training

Neural network training is to find the set of neural weights that provide the best fit between a set of network output and a set of provided output, given the same set of input data. Typically, these problems are of minimizing the sum of least squares errrors

$$\underset{r\in\mathbb{R}^K}{\text{minimize}}\ \frac{1}{2}\sum_{t=1}^{m} e^2(t), \tag{3-34}$$

with error e being defined as

$$e(t) = \left\| J(i_t) - \tilde{J}(i_t, r_t) \right\|. \tag{3-35}$$

where $\tilde{J}(i_t, r_t)$ is the approximation function and $J(i)$ the exact values. Throughout this thesis, $\|\cdot\|$ stands for the Euclidean norm, given by $\|\cdot\| = \sqrt{x^{\mathrm{T}} x}$.

We can further define an error function

$$\xi_t = \frac{1}{2} e^2(t), \tag{3-36}$$

and rewrite Eq. (3-39) as

$$\underset{r\in\mathbb{R}^K}{\text{minimise}}\ \sum_{t=1}^{m} \xi(t). \tag{3-37}$$

Given a state sequence $\{i_t \mid t = 0, 1, \ldots, m\}$ and an observation sequence $J_1, J_2, \ldots, J_m$, to solve the least square problem defined by Eq. (3-36) and (3-37), we can use *linear least-square* (LLSQ) algorithm for linear approximation, and use a *gradient descent* algorithm or *Newton's method* for non-linear approximation. The concern here is that methods above have to process the entire data set of the pair of i_t and J_t before updating r, and the size of data can make computation costly because of computing correlation function and matrix inversion. On the other hand, in real-time operation observation of J_t may only occur once per time increment. Consequently, we prefer an incremental method for the least square problem so that

$$r \leftarrow r + \sum_{t=1}^{m} \Delta r_t \tag{3-38}$$

where Δr is an incremental correction of parameter value. This is a distinctive property of *least-mean-square* (LMS) algorithm (Widrow and Hoff, 1960). To show the incremental update in LMS, let us first consider an function approximation built on a linear neuron

$$\tilde{J}(i,r) = \sum_{j=1}^{K} r(j)\phi_j(i) + b \ . \tag{3-39}$$

If we take the partial derivative of the squared error with respect to the synaptic weights r and biases b at the t^{th} iteration, we have

$$\frac{\partial \xi(t)}{\partial r_t(j)} = e(t)\frac{\partial e(t)}{\partial r_t(j)} \quad \text{for } j = 1,2,\ldots,K \ ,$$

and

$$\frac{\partial \xi(t)}{\partial b_t} = e(t)\frac{\partial e(t)}{\partial b_t} .$$

Next, we look at the partial derivertives in respect to the error e_t.

$$\frac{\partial e(t)}{\partial r_t(j)} = \frac{\partial\left(J(i) - \tilde{J}(i,r_t)\right)}{\partial r_t(j)} = \frac{\partial}{\partial r_i(j)}\left(J(i) - \left(\sum_{j=1}^{K} r_t(j)\phi_j(i) + b_t\right)\right),$$

which can be simplied as

$$\frac{\partial e(t)}{\partial r_t(j)} = -\phi_j(i) ,$$

and

$$\frac{\partial e(t)}{\partial b_t} = -1 .$$

Then, the change of the weight and the bias are calculated according to the *delta rule*

$$\Delta r_{ji}(t) = -\eta \frac{\partial \xi(t)}{\partial r_{ji}(t)} ,$$

which gives:

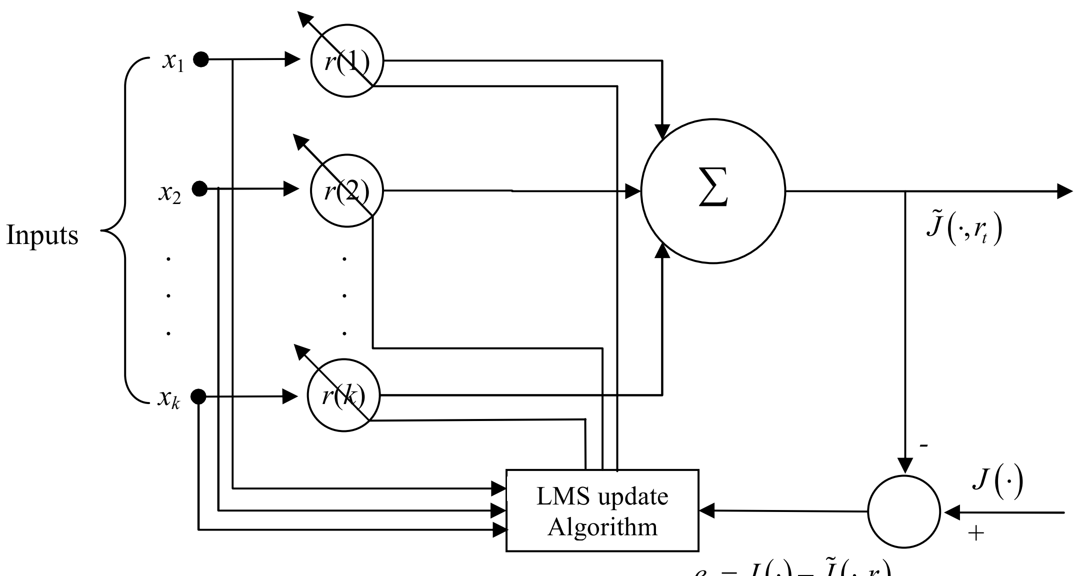


Fig. 3-11 Information flow graph representation of the LMS algorithm; estimated output $\tilde{J}$ from the neural network is measured against to the ideal (target) output J ; the error between the estimated and ideal output is processed by the LMS algorithm and then propagated backward to the neural network to correct neural weights *r*.

$$\Delta r_t = \eta e(t) \phi'(i), \tag{3-40}$$

and

$$\Delta b_t = \eta e(t), \tag{3-41}$$

where η is a learning rate that serves as a measure of the *memory* of the LMS algorithm.

Finally, weights and bias are updated through

$$r_{t+1} = r_t + \Delta r_t, \tag{3-42}$$

and

$$b_{t+1} = b_t + \Delta b_t. \tag{3-43}$$

It is customary to assign arbitrarily initial values to r_0 and b_0. Proof on convergence of LMS can be found in Haykin (Section 3.5, 1999). A graphical representation of LMS algorithm is shown in Fig. 3-11.

Comparing to other incremental methods to adjust functional parameters, such as Kalman filtering, the LMS is simple and robust. It does not require memory of a set of past estimations or calculating matrix inversion, thus being efficient in computation. It does not require knowledge of the statistics of the environment, thus being model free and robust. Comparing to method of steepest decent, which produced a well-fined tragectory in parameter space, the LMS algorithm produced a stochastic tragectory. This means that the main drawback of the LMS is the rate of convergence in terms of the number of iterations required.

As for non-linear approximation built on multi-layer neural networks, the usual method for training network weights is *back-propagation* (Werbos, 1974). Similar to LMS, it looks for the minimum of the error function in weight space by incrementally applying a correction Δr_t to the synaptic weight r_t. The denomination of this algorithm reflects the backward pathway of error information as in contrast to the forward propagation of input information. Back-propagation can be broadly seen as an extended LMS to non-linear neural network.

We consider a neuron j in a multi-layer non-linear neural network that maps $[x, r]$ to $\tilde{J}(x,r)$, where x denotes neuron input vector. The objective for network training is the same as stated in Eq.(3-36) and (3-37), and the correction $\Delta r_{ji}(t)$ is calculated by the *delta rule*:

$$\Delta r_{ji}(t) = -\eta \frac{\partial \xi(t)}{\partial r_{ji}(t)}. \tag{3-44}$$

For the consistence with Eq. (3-32) which defines a layered network, we use $y_j(t)$ to denote the scalar output of non-linear neuron j, and consequently define that

$$\xi(t) = \frac{1}{2}\sum_{j\in C} e_j^2(t) = \frac{1}{2}\sum_{j\in C}\left(J(x(t)) - y_j(t)\right)^2, \tag{3-45}$$

where J is the exact output and C is a set of neurons. We further define that:

$$e(t) = J(x(t)) - y_j(t), \tag{3-46}$$

$$y_j(t) = \varphi_j(v_j(t)), \tag{3-47}$$

$$v_j(t) = \sum_{i=0}^{m} r_{ji} y_i(t). \tag{3-48}$$

The derivative of error function in respect to weight can be written in chain rule as

$$\frac{\partial \xi(t)}{\partial r_{ji}(t)} = \frac{\partial \xi(t)}{\partial e_j(t)} \frac{\partial e_j(t)}{\partial y_j(t)} \frac{\partial y_j(t)}{\partial v_j(t)} \frac{\partial v_j(t)}{\partial r_{ji}(t)}. \tag{3-49}$$

Differentiating both sides of Eq. (3-45) with respect to $e_j(t)$, we get

$$\frac{\partial \xi(t)}{\partial e_j(t)} = e_j(t). \tag{3-50}$$

Differentiating both sides of Eq. (3-46) with respect to $y_j(t)$, we get

$$\frac{\partial e_j(t)}{\partial y_j(t)} = -1. \tag{3-51}$$

Then, we differentiate both sides of Eq.(3-47) with respect to $v_j(t)$ and get

$$\frac{\partial y_j(t)}{\partial v_j(t)} = \varphi'(v_j(t)), \tag{3-52}$$

where the use of prime signifies differentiation with respect to the arguement. Finally, differentiatnig Eq. (3-48) with respect to $r_{ji}(t)$ yields

$$\frac{\partial v_j(t)}{\partial r_{ji}(t)} = y_i(t). \tag{3-53}$$

Substituting Eq. (3-50) — (3-53) to Eq. (3-49), we get

$$\frac{\partial \xi(t)}{\partial r_{ji}(t)} = -e_j(t)\varphi_j'(v_j(t))y_i(t). \tag{3-54}$$

Accordingly, the use of Eq. (3-54) in (3-44) yields

$$\Delta r_{ji}(t) = \eta \delta_j(t) y_i(t), \tag{3-55}$$

where the *local gradient* $\delta(t)$ is defined by

$$\delta_j(t) = e_j(t)\varphi_j'(v_j(t)). \tag{3-56}$$

Neural weights are updated in the same manner as Eq. (3-44). The bias in the expression of back-propagation has been integrated into r_{j0} (corresponding to the fixed input $y_0 = +1$).

This far, we have discussed the learning paradigms and the process for learning signals to propagate through neural networks to update neural weights. To complete the learning process, a learning target is required. In supervised learning the targets are the corresponding exact outputs, and training can be either performed by using batch-learning techniques such as Newton's method, or by using incremental methods such as LMS and back-propagation algorithm. In reinforcement learning, we do not have a source of exact outputs, otherwise dynamic programming would just meet the need of adaptive control, albeit at a computational burden. In the following section, we discuss a reinforcement learning technique that allows a system to learn from its own interactions with the process, and updates approximation incrementally over time.

3.4.4 Temporal-difference (TD) learning

Temporal difference (TD) learning, originally proposed by Sutton (1988), is a method for approximating long-term future cost as a function of current state. The TD method is central to reinforcement learning in that it does not require a source to provide exact learning target; rather, it learns from observing actual state transition cost and then updates approximation parameters according to the *difference* between estimation and the actual observation. In functional approximation to J, the TD method solves a stochastic prediction problem in which experience comes in observation-outcome sequences of the form i_0, i_1, …, i_m, J, where each i_t is a vector of observations available at time t in the sequence, and J is the outcome of the sequence. For each observation-outcome sequence, the learner produces a corresponding sequence of approximations $\tilde{J}(i_t)$. The approximations are also based on a vector of modifiable parameters or weights, r. Since approximation function $\tilde{J}(i_t)$ depends both on i_t and r, and it can be rewritten as $\tilde{J}(i_t, r)$.

Let a linear approximation function $\tilde{J}(i,r)$ defined by Eq. (3-39), by building bias b into $r(0)$ corresponding to fixed input $\phi_0 = +1$, we have

$$\tilde{J}(i,r) = \sum_{j=0}^{K} r(j)\phi_j(i) .$$

Furthermore, we define a projection operator $\prod$ as

$$\prod J = \arg \min_{\tilde{J} \in \left\{ \Phi r' \middle| r \in \mathbb{R}^K \right\}} \left\| J - \tilde{J} \right\|, \tag{3-57}$$

where the norm || · || can be defined by letting

$$\|J\| = \langle J, J \rangle^{1/2},$$

where $\langle \cdot, \cdot \rangle$ is the inner product. The approximation function $\tilde{J}$ (·, *r*) can be seen as an orthogonal projection of *J* on $\left\{ \Phi r' \middle| r \in \mathbb{R}^K \right\}$ with respect to inner product $\langle \cdot, \cdot \rangle$. The projection ∏*J*, therefore, is a natural approximation to *J*, given the fixed sets of basis functions Φ. In particular, ∏ *J* is the solution to the least-squares problem of

$$r^* = \arg\min_r \sum_{i \in S} \left[J(i) - \tilde{J}(i, r) \right]^2 \quad . \tag{3-58}$$

Proof of ∏*J* defined by Eq. (3-57) as a solution to Eq. (3-58) is provided in Van Roy (Theorem 3.9, 3.10, 3.11, 1998) and in Appendix 3.A of this thesis (Lemma 3.A.6).

To update the approximation in TD learning is then to update the parameter vector *r* so that Eq. (3-58) is solved. Furthermore, we assume that *r* is updated only once for each complete observation-outcome sequence and thus does not change during a sequence. For each observation, an increment Δ*r* is determined, and after a complete sequence has been processed, *r* is changed by all the sequence's increments:

$$r \leftarrow r + \sum_{t=0}^{m} \Delta r_t \ .$$

From the discussion in Section 3.4.3, we can either employ gradient method, such as deepest decent or Newton's method, to update *r* if we wait until the entire process of i_t is completed and *J*(*i*) becomes available, or we can use LMS or back-propagation if a desired response is provided at each time increment. In reinforcement learning, we cannot wait for the procession of the entire state sequence before updating *r*. Instead, the TD approach represents the error $J - \tilde{J}_t$ as a sum of changes in approximation, that is, as

$$J - \tilde{J}_t = \sum_{k=t}^{m} \left(\tilde{J}_{k+1} - \tilde{J}_k \right) \text{ where } \tilde{J}_{m+1} \overset{\text{def}}{=} J \ . \tag{3-59}$$

Function parameter *r* is updated incrementally as

$$\begin{aligned} r \leftarrow r + \sum_{t=1}^{m} \eta\left(J - \tilde{J}_t\right)\nabla_r \tilde{J}_t &= r + \sum_{t=1}^{m} \eta \sum_{k=t}^{m} \left(\tilde{J}_{k+1} - \tilde{J}_k\right)\nabla_r \tilde{J}_t \\ &= r + \sum_{k=1}^{m} \eta \sum_{t=1}^{k} \left(\tilde{J}_{k+1} - \tilde{J}_k\right)\nabla_r \tilde{J}_t \ , \\ &= r + \sum_{t=1}^{m} \eta\left(\tilde{J}_{t+1} - \tilde{J}_t\right)\sum_{k=1}^{t} \nabla_r \tilde{J}_k \end{aligned} \quad (3\text{-}60)$$

where the incremental correction Δr_t is expressed as

$$\Delta r_t = \eta\left(\tilde{J}_{t+1} - \tilde{J}_t\right)\sum_{k=1}^{t} \nabla_r \tilde{J}_k \ . \quad (3\text{-}61)$$

Because each Δr_t depends only on a pair of successive approximation and on the sum of all past values for $\nabla_r \tilde{J}_t$, this process reduces memory demand substantially. We refer to the process given by Eq. (3-61) as the TD(1) method.

In response to the difference between two successive approximations, TD(1) is a special case in which some or all of the preceding approximations are altered to an equal extent. In other cases, we may prefer to assign greater weight to more recent approximations. This leads us to a generalisation of TD procedure by introducing an exponential weighting factor λ so that for $0 \leq \lambda \leq 1$, we have

$$\Delta r_t = \eta\left(\tilde{J}_{t+1} - \tilde{J}_t\right)\sum_{k=1}^{t} \lambda^{t-k} \nabla_r \tilde{J}_k \ . \quad (3\text{-}62)$$

Note that for $\lambda = 1$, Eq. (3-62) is equivalent to the TD(1) procedure. For $\lambda<1$, TD(λ) produces weight changes different from the Widrow-Hoff procedure where exact learning targets are known. The difference is the greatest in the case of TD(0) where $\lambda = 0$. This is because weight increment in TD(0) is determined only by its effects on the approximation with the most recent observation

$$\Delta r_t = \eta\left(\tilde{J}_{t+1} - \tilde{J}_t\right)\nabla_r \tilde{J}_t \, . \quad (3\text{-}63)$$

It can be seen that TD(0) has the same learning mechanism as LMS, but with different errors. It is easy to verify that that the TD(1) procedure converges to the true cost-to-completion *J*, we here prove that the TD(λ) approximation in general coverges asysmptotically to the true value.

Tsitsiklis and Van Roy (1997) generalised the convergence theory for TD(λ) algorithms within the domain of infinite-horizon and finite-state dynamic programming problems with discounted cost and linear cost function approximation. Before the formal statement of assumptions and convergence theory, let us define a TD(λ) operator on a discrete time system by

$$\left(T^{(\lambda)}J\right)(i) = (1-\lambda)\sum_{m=0}^{\infty}\lambda^m E\left[\sum_{t=0}^{m}\alpha^t g(i_t,i_{t+1}) + \alpha^{m+1}J(i_{m+1}) \mid i_0 = i\right] \quad (3\text{-}64)$$

where λ ∈ (0,1). In the case where λ = 1, we define the operator as:

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

which is identical to Eq. (3-8); in the case where λ = 0, we define the operator as:

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

For $i \in X$, we introduce weighted function spaces $L_2(X, D)$ to denote the set of vectors

$$\left\{J \in \mathbb{R}^n \middle| \|J\|_D = \sqrt{J'DJ} < \infty\right\}.$$

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

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

and $\pi(i)$ stands for the steady-state probability of visiting state i. We further denote the expectation with respect to the steady state distribution as $E_0[\cdot]$. The outstanding assumptions for this convergence theorem are:

Assumption 3-3 *(a) The Markov chain has steady-state probability $\pi(1)$, ..., $\pi(n)$ which are positive, that is,*

$$\lim_{t\to\infty} P\{i_t = j | i_0\} = \pi(j) > 0,\ \forall i_0, j,$$

and

$$\pi' P = \pi'.$$

(b) The one-step costs $g_t(i_t, i_{t+1})$ *satisfy*

$$E_0\left[g^2\left(i_t, i_{t+1}\right)\right] < \infty .$$

The next assumption ensures that the basis functions (or the feature-extraction functions) are linearly independent and do not grow too fast.

Assumption 3-4 *(a) The matrix Φ given by Eq. (3-29) has full rank.*

(b) Each basis function satisfies

$$E_0\left[\phi_j^2\left(i\right)\right] < \infty \text{ for } j = 0,1,...,K .$$

The next assumption essentially requires that the Markov chain has a certain "degree of stability."

Assumption 3-5 *There exists a function* $f\colon X \to \mathbb{R}^+$ (*nonnegative real numbers*) *satisfying the following requirements:*

(a) For all i_0 *and* $m \geq 0$,

$$\sum_{\tau=0}^{\infty}\left\| E\left[\phi(i_\tau)\phi'(i_{\tau+m})|i_0\right] - E_0\left[\phi(i_t)\phi'(i_{t+m})\right]\right\| \leq f\left(i_0\right),$$

and

$$\sum_{\tau=0}^{\infty}\left\| E\left[\phi(i_\tau)g\left(i_{\tau+m}, i_{\tau+m+1}\right)|i_0\right] - E_0\left[\phi(i_t)g\left(i_{t+m}, i_{t+m+1}\right)\right]\right\| \leq f\left(i_0\right).$$

(b) For any $n > 1$, *there exists a constant* ω_n *such that for all* i_0, t,

$$E\left[f^n\left(i_t | i_0\right)\right] \leq \omega_n f^n\left(i_0\right).$$

The final assumption places standard constraints on the sequence of stepsizes.

Assumption 3-6 *The learning rate, (or stepsize),* η_t *are positive, non-increasing, and predetermined, and moreover, satisfies*

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

The convergence theorem can then be stated as:

Theorem 3.2 (Tsitsiklis and Van Roy, 1997) *Under Assumptions 3-3, 3-4, 3-5, and 3-6, the followings hold:*

(a) *The exact J values are in $L_2(X, D)$.*

(b) *For any $\lambda \in [0,1]$, the TD(λ) algorithm with linear approximation function converges with probability* 1.

(c) *The limit of convergence r^* is the unique solution of the equation*

$$\Pi T^{(\lambda)}\left(\Phi r^*\right) = \Phi r^* .$$

(d) *Furthermore, r^* satisfies*

$$\left\|\Phi r^* - J\right\|_D \le \frac{1-\alpha\lambda}{1-\alpha}\left\|\Pi J - J\right\|_D$$

The proof of Theorem 3.2 is originally provided in Tsitsiklis and Van Roy (1997) and is again provided in Appendix 3.A using the notations and definitions of this study. It can be shown that Assumptions 3-3(b), 3-4(b), and 3-5 are automatically true whenever an irreducible aperiodic Markov Chain with a finite state space is considered.

The essence of this analysis is to write the TD(λ) algorithm as

$$r_{t+1} = r_t + \eta_t\left(Ar_t + c\right), \tag{3-65}$$

where η_t is a positive learning rate satisfying Assumption 3-7. Convergence is obtained at

$$Ar + c = 0 , \tag{3-66}$$

where A and c are given by

$$A = \Phi' D\left(P - I\right)\sum_{m=0}^{\infty}\left(\alpha\lambda P\right)^m \Phi , \tag{3-67}$$

$$c = \Phi' D\sum_{m=0}^{\infty}\left(\alpha\lambda P\right)^m \bar{g} , \tag{3-68}$$

where D is the diagonal matrix with diagonal entries $\pi(i)$, $i = 1,\ldots, n$, and $\bar{g}$ is the vector with components

$$\bar{g}\left(i\right) = \sum_{j=1}^{n} p_{ij} g\left(i, j\right) . \tag{3-69}$$

We can have an additional insight in Eq. (3-65) by employing the generalised form of TD(λ) expressed by Eq. (3-62), and rewrite Eq. (3-65) as

$$\begin{aligned} r_{t+1} &= r_t + \eta_t\left(\tilde{J}_{t+1} - \tilde{J}_t\right)\sum_{k=1}^{t}\lambda^{t-k}\nabla_r \tilde{J}\left(i_k\right) \\ &= r_t + \eta_t d_t \sum_{k=1}^{t}\lambda^{t-k}\phi\left(i_k\right), \end{aligned}$$

where we use

$$d_t = \tilde{J}_{t+1} - \tilde{J}_t \tag{3-70}$$

to denote the temporal difference at time *t*. For the special case where λ = 0, we have

$$r_{t+1} = r_t + \eta_t d_t \phi\left(i_t\right). \tag{3-71}$$

Theorem 3.2 provides the theoretical justification for incremental adjustment to *r* in linear function approximation using TD(λ) learning. This method is easy to implement and efficient in computing. There are, however, two concerns about this approach.

The first concern is Assumption 3-6, which requires stepsize η_t diminishing to zero. A deterministic and diminishing stepsize slows down the rate of convergence, as shown by Nedić and Bertsekas (2003), who proposed a λ-least square policy evaluation method (λ-LSPE) as an alternative to TD(λ) in approximating the exact *J* values. By bringing in least square method to update *r* at each iteration, the λ-LSPE method guarantees convergence while adopting a constant stepsize η = 1. The convergence by using λ-LSPE is faster than TD(λ) in terms of number of iterations. Nevertheless, λ-LSPE approach requires use of a correlation function and computes matrix inversion. This will be computationally costly if the state space is large. Similar arguments apply to the least square temporal difference method (LSTD) by Boyan (2002) which solves directly the system of equations that characterises the convergence of TD(λ), using all the information available from simulation.

The second concern is that if an online TD(λ) method does not sample states with the frequencies of the Markov chain, it does not always converge. This has been evidenced by specific case studies shown in Bertsekas and Tsitsiklis (Example 6.12, 1996), and Tsitsiklis and Van Roy (Section 9, 1997).

Finally, we notice that the implication of TD(λ) method so far only pertains to *single step* value iteration, which means that each iteration only involves a single step state transition from i_t to i_{t+1}. In online traffic signal control, we may have information about further arriving traffic from upstream detectors, which may cover several discrete time steps. To utilise the detected information or to simulate the process further into time, it may involve a multi-step transitions. This leads to TD(λ) in multi-step value iteration, which is discussed in the next section.

3.4.5 TD(λ) and multi-step value iteration

Multi-step value iteration involves multiple state transitions in a single iteration. In particular, for $M \geq 1$, let us consider the M-transition Bellman's equation

$$J(i) = E\left[\sum_{t=0}^{M-1} \alpha^t g(i_t, i_{t+1}) + \alpha^M J(i_M) \big| i_0 = i\right],\ i = 1,2,...,n$$

The value iteration method corresponding to this modified problem is

$$J_{t+1}(i) = E\left[\sum_{k=t}^{t+M-1} \alpha^k g(i_k, i_{k+1}) + \alpha^M J_t(i_{t+M-1}) \big| i_0 = i\right],\ i = 1,2,...,n, \tag{3-72}$$

Using the definition of temporal difference, we solve this problem by finding

$$r_{t+1} = \arg\min_r \sum_{m=0}^{t} \left(J(i_m, r) - \tilde{J}_t(i_m, r_t) - \sum_{k=m}^{m+M-1} \alpha^{k-m} d_k(i_k, i_{k+1}) \right),\ t = 0,1,... \tag{3-73}$$

Using the approximation function $\tilde{J}(\cdot, r)$ defined by Eq. (3-25), the incremental gradient version of the iteration Eq. (3-73) is given by

$$r_{t+1} = r_t + \eta_t \phi(i_t) \sum_{k=t}^{t+M-1} \alpha^{k-t} d_k(i_k, i_{k+1}),\ t = 0,1,.... \tag{3-74}$$

This incremental approach pertains to neither TD(0) nor any specific TD(λ). Bertsekas *et al* (2004) consider this method as an intermediate between TD(0) and TD(λ) — it is closest to TD(0) for small M, and to TD(1) for large M. Using Theorem 3.2 and its assumptions, we can verify that Eq. (3-74) converges.

Eq. (3-74) is the method adopted in this study to train neural networks. It can be directly implemented to back-propagation. It offers an opportunity of looking forward into time, utilising information about the future events, rather than looking only backward through time. Past information is built into the evolving functional parameters.

3.4.6 Summary

In Section 3.4, we introduced linear and non-linear approximations to exact *J* values. We generalised the concept of functional approximation using neural networks. Neural networks, by choosing appropriate architecture, can exhibit either linearity or non-linearity. The training of neural networks requires machine-learning techniques. Supervised and unsupervised learning are the two common learning paradigms. Without a source of exact output to correct the neural network output, our discussion focused on unsupervised learning, and on reinforcement learning in particular. Reinforcement learning adopts a trial-and-error method so that the learning agent accumulates knowledge by observing its own behaviour. The learning process generates a correction every time when an observation of behaviour becomes available. This correction is propagated back to the learning agent so that the parameters of the approximation functions are adjusted incrementally. The LMS is a common incremental method for linear approximation function and back-propagation for non-linear multi-layer neural networks. Back-propagation can be seen as a generalised LMS method. The temporal difference learning is the key to reinforcement learning. We showed that under a few assumptions, the TD(λ) process converges with probability of 1. Of particular interest to us is using TD(λ) in multi-step value iteration, in which a single iteration may involve multiple state transitions. We showed that TD with multi-step value iteration is a special case that can be seen as an intermediate between TD(0) and TD(1).

Unsupervised learning is a general concept, and therefore is not limited only to TD(λ) method. In the next section, we introduce perturbation learning as another method of formulating unsupervised learning.

3.5 Perturbation Learning

In Section 3.4, we discussed about linear approximation built on neural networks and trained by temporal difference learning method TD(λ). We discuss perturbation learning here as an alternative learning method to adjust the linear approximation function $\tilde{J}(\cdot,r)$. The function parameters are estimated by perturbing system state with partial increment so that the partial derivatives are calculated numerically. This method is motivated by the structural property of the $J(\cdot)$ function. Papadaki and Powell (2003) used this approach to solve batch service problems, which share similarity with the traffic signal control problem.

3.5.1 Monotonicity of the value function

We begin the discussion here with the structural properties of the $J(\cdot)$ function. Let i denote system state, w information of exogenous process, and μ control policy, we have

$$u = \arg\min_{u\in U(\mu)} \underset{w}{E}\left\{ g(i,u) + \alpha J\left(j \middle| i,w,u\right)\right\} \quad i,j \in X . \tag{3-75}$$

A generic representation of state transition can be written as

$$i_{t+1} = f\left(i_t,u_t\right) + w_t . \tag{3-76}$$

where function $f(\cdot)$ denotes the vector that gives the state after a decision is implemented. Since the process of w_t is exogenous and random, a stochastic version of Eq. (3-76) can be written as

$$\Pr\left(i_{t+1} \middle| i_t,u_t\right) J\left(i_{t+1}\right) = \Pr\left(w_t\right) J\left(f\left(i_t,u_t\right) + w_t\right) . \tag{3-77}$$

An important structural property of the $J(\cdot)$ function is its monotonicity. Puterman (Proposition 4.7.3, page 106-107, 1994) establishes the conditions for non-decreasing (non-increasing) monotonicity of $J(\cdot)$. Papadaki and Powell (2007) extended the monotonicity results to partially ordered multidimensional case. Before preceding the discussion on the conditions, we need the following definitions.

We define partial ordering operator $\preceq$ or $\succeq$ on the N-dimensional set X. We define $i \preceq j$ *or* $i \succeq j$ for $i,j \in X$, if for all $k \in \{1,2,...,N\}$ we have $i(k) \le j(k)$, or $i(k) \ge j(k)$.

We further define a real-valued function $J : X \to \mathbb{R}$ as partially non-decreasing or non-increasing if for all $i^+, i^- \in X$ such that $i^+ \succeq i^-$, we have $J(i^+) \geq J(i^-)$ or $J(i^+) \leq J(i^-)$.

Furthermore, given that if for some $k \in \{1,2,...,N\}$ we have $j(k) < J(i, u)(k)$, then $\Pr(j|i,u) = 0$, since w_t is assumed non-negative. We only need to take expectation over state j that satisfies $j \succeq f(i,u)$ in the following equations concerning state transition.

To prove our main result we need the following lemma:

Lemma 3.1 *Suppose that $J_{t+1}(i)$ is partially non-decreasing (non-increasing) in X, and that for $\Delta i \succeq 0$, $f(i+\Delta i,u) \succeq f(i,u)$, $\forall u \in U$, then we have,*

$$\sum_{j \in X, j \succeq f(i+\Delta i,u)} \Pr(j|i+\Delta i,u) J_{t+1}(j) \geq \sum_{j \in X, j \succeq f(i,u)} \Pr(j|i,u) J_{t+1}(j), \tag{3-78}$$

(*where the inequality is reversed in the non-increasing case*).

Proof: For $j \succeq f(i,u)$ and by Eq. (3-77) we have

$$\Pr(j|i,u) J(j) = \Pr(w) J(f(i,u)+w). \tag{3-79}$$

Using (3-79), (3-78) becomes

$$\sum_w \Pr(w) J_{t+1}(f(i+\Delta i,u)+w) \geq \sum_w \Pr(w) J_{t+1}(f(i,u)+w). \tag{3-80}$$

The above holds under the assumption that J_{t+1} and $f(\cdot)$ are partially non-decreasing respectively. **q.e.d.**

The following theorem states sufficient conditions for partial monotonicity of the J function in our study.

Theorem 3.3 *Suppose the following conditions hold:*

(a) For $\Delta i \succeq 0$ we have $f(i+\Delta i,u) \succeq f(i,u)$, $\forall u \in U$.

(b) The one period cost function $g(\cdot)$ is partially non-decreasing (non-increasing) in $i \in X$ for all $u \in U$, $t = 0, 1, \ldots, m-1$.

(c) The terminal cost $h(i_m)$ is partially non-decreasing (non-increasing) in $i \in X$.

Then value function J is partially non-decreasing (non-increasing) in $i \in X$.

Proof: Under condition (c) the results hold for $J(i_m) = h(i_m)$. For any J, by Eq. (3-79), we have

$$J(i) = \min\left\{ g(i,u) + \alpha \sum_w \Pr(w) J\left(f(i,u) + w \right) \right\}.$$

Given that the decision vector space is finite, there exists $u_t^+ \in U$ that attains the above minimum for state $i = i^+$. Thus, the J function can be written as

$$J\left(i^+\right) = g\left(i^+, u^+\right) + \alpha \sum_w \Pr(w) J\left(f\left(i^+, u^+\right) + w \right).$$

For $i^+ \succeq i^-$, and due to condition (a) and (b) and Eq. (3-80), we have

$$\begin{aligned} J\left(i^+\right) &\geq g\left(i^-, u^+\right) + \alpha \sum_w \Pr(w) J\left(f\left(i^-, u^+\right) + w \right) \\ &\geq \min_{u \in U} \left\{ g\left(i^-, u\right) + \alpha \sum_w \Pr(w) J\left(f\left(i^-, u\right) + w \right) \right\} \\ &= J\left(i^-\right). \end{aligned}$$

$J\,(i)$ is therefore partially non-decreasing in i. **q.e.d.**

Theorem 3.3 is derived from Puterman (1994) and Papadaki and Powell (2007). The former work established sufficient conditions for monotonicity of value function involving scalar state variable, and the latter extended the conditions to include partially-ordered vector state variable.

The monotonicity of J function for a traffic signal control problem is discussed in Chapter 4 after the introduction of system dynamics. In the rest of this section, we show the general approach to establish perturbation learning.

3.5.2 Perturbation of system state

Consider a linear approximation function $\tilde{J}(i,r)$ defined by Eq. (3-39), and let $\Delta i(k)$ be a N-dimensional vector of zeroes except for a unit increment in the kth element, perturbation learning computes

$$\Delta r_t(k) = \frac{\hat{J}\left(i_t + \Delta i(k)\right) - \hat{J}\left(i_t\right)}{\left\| \Delta i(k) \right\|}, \tag{3-81}$$

where

$$\hat{J}(i_t) = \min_{u_t \in U(\mu)} \underset{w}{E}\left\{g(i_t, w_t, u_t) + \alpha\tilde{J}\left(f(i_t, u_t) + w_t, r_t\right)\right\}.$$

We then smooth to obtain an updated estimate of the function parameter

$$r_{t+1}(k) = (1-\eta_t)r_t(k) + \eta_t \Delta r_t(k), \text{ for } k = 1, 2, ..., N, \tag{3-82}$$

where η_t is a sequence of diminishing stepsizes that satisfy Assumption 3-6.

The perturbation learning overall is simple for implementation. A drawback of this approach is that the computational demand increases substantially as the dimension of state vector *i* increases. This is because $\Delta r(k)$ is computed numerically at each time, which is unlike the temporal difference (TD) learning that updates parameter vector *r* simultaneously. For the same reason, extending the perturbation learning to non-linear approximation becomes even more difficult. Papadaki and Powell (2003) in studying a simple batch service problem compared linear and non-linear function forms for approximation, and found out that non-linear approximation only works better after a number of iterations, suggesting that linear approximation is more cost-effective for problem of large state space.

3.5.3 Summary

Perturbation learning adjusts parameters of an approximation function by perturbing system state with a partial increment. The adjustment of each parameter is obtained by numerically calculating the partial gradient corresponding to the partial increment. It offers an alternative in establishing unsupervised learning for the purpose of improving approximation.

3.6 Discussion

In Chapter 3 we introduced approximate dynamic programming (ADP) as the theoretical framework for developing a new adaptive traffic signal controller. The ADP concept is a practical and evolutionary substitution to classic backward dynamic programming (DP). The DP solution, although being the only exact solution for optimisation over time, usually becomes intractable under high dimensionality, and impractical for real-time implementation. ADP seeks to reduce dimensionality and computation requirement by replacing a look-up table of exact *J* values with an approximation function. The ADP concept is general, and accommodates different forms of approximation function, including linear and non-linear forms. These approximations can be united under the general definition of artificial neural networks, and in particular, the ability of non-linear neural works in functional approximation is supported by the universal approximation theorem (Theorem 3.1). The general method for training neural networks is back-propagation, which by implication propagates the error signal backwards in a layer-by-layer manner to update network parameters (or synaptic weights in neuron terms). Back-propagation can be seen as a generalised least-mean-square (LMS) method, which is of particular interest to online operation, as it updates functional parameters incrementally.

The training of neural networks requires machine-learning techniques. Supervised learning and unsupervised learning are the major learning paradigms. The latter is of particular interest to real-time control, as it does not require a source of exact output for the neural networks to match. Reinforcement learning is an important concept in the paradigm of unsupervised learning. It uses the mathematical formalism of dynamic programming and learns from its own interaction with environment. A reinforcement learning technique that tracks the difference between its own estimations and new observations of state transition pertains to the temporal difference (TD) method. We have shown in this chapter that under a few assumptions, linear approximation function trained by TD converges with probability of 1. Apart from the TD method, we introduced perturbation learning as an alternative method to establish unsupervised learning. This approach calculates partial gradient of a linear approximation function by perturbing system state with an artificial increment. This approach is motivated by the monotonicity property of the J function.

Before leaving this chapter, we reiterate here the discussion on the two concerns related to the TD method that were identified in Section 3.4.4. The first concern is about the diminishing stepsize and the second about the Markov process. From Theorem 3.2, we notice that a diminishing stepsize is essential for the convergence of TD. However, in practice, we may not know the optimal stepsize easily (Chapter 6, Powell, 2007), and furthermore, if stepsize approaches zero too rapidly, the learning process stops prematurely. On the other hand, if we start with a large initial stepsize, the neural networks may overshoot in learning. Because the focus of this study is to develop a system that constantly adapts to changes in prevailing traffic and optimises performance over time, we may be satisfied with a functional approximation that has acceptable error bound instead of convergence. It can be shown that the error of linear function approximations trained by TD is indeed bounded (Lemma 3.A.6, Appendix 3.A), and is not conditional on particular stepsize property. In fact, if prevailing traffic varies over time, there is little to be achieved by seeking immediate convergence. This motivates us to adopt a constant but cautious stepsize for TD method.

The second concern is that if state is not sampled in a frequency natural to Markov process, convergence will not be achieved. To eliminate the possibility of divergence, any arbitrary policies that determine state transition should be avoided. In this study, we only discuss control policies that are greedy with respect to the value function.

In the next Chapter, we introduce system dynamics for both isolated and network traffic signal control. ADP algorithms will be established for both cases.

Appendix 3.A Convergence of TD(λ) with Linear Function Approximation

We start proving Theorem 3.2 with the fundamental lemma on Markov chains. Here, for $i \in X$, we use $L_2(X, D)$ to denote the set of vectors

$$\left\{ J \in \mathbb{R}^n \left| \|J\|_D = \sqrt{J'DJ} < \infty \right. \right\},$$

where D is the diagonal matrix with diagonal entries $\pi(i)$, $i = 1,\ldots, n$, and $\pi(i)$ is the steady-state probability of being state i.

Lemma 3.A.1 *Under Assumption 3-3(a), for any* $J \in L_2(X, D)$*, we have*

$$\|PJ\|_D \le \|J\|_D .$$

Proof: The proof involves Jensen's inequality and Tonelli's theorem:

$$\begin{aligned}
\|PJ\|_D^2 &= J'P'DPJ \\
&= \sum_{i=1}^{n} \pi(i) \left(\sum_{j=1}^{n} p_{ij} J(j) \right)^2 \\
&\le \sum_{i=1}^{n} \pi(i) \sum_{j=1}^{n} p_{ij} J^2(j) \\
&= \sum_{j=1}^{n} \sum_{i=1}^{n} \pi(i) p_{ij} J^2(j) \\
&= \sum_{i=1}^{n} \pi(i) p_{ij} J^2(j) \\
&= \sum_{i=1}^{n} \pi(i) J^2(j) \\
&= \|J\|_D^2
\end{aligned}$$

Lemma 3.A.2 *Under Assumption 3-4 (a) and (b), J(i) is well-defined and finite,* $\forall i \in X$. *Furthermore, J is in L₂(X, D), and*

$$J = \sum_{t=0}^{\infty} (\alpha P)^t \, \bar{g} .$$

Proof: If the Markov chain starts in steady state, it remains in steady-state, and therefore

$$E_0 \left[\sum_{t=0}^{\infty} \alpha^t g^2(i_t, i_{t+1}) \right] = \frac{1}{1-\alpha} E_0 \left[g^2(i_t, i_{t+1}) \right] < \infty ,$$

Because $|g(i_t, i_{t+1})| \le 1 + g^2(i_t, i_{t+1})$, it follows that

$$\sum_{i\in X}\pi(i)E\left[\sum_{t=0}^{\infty}\alpha^t\left|g(i_t,i_{t+1})\right|\middle|i_0=i\right]=E_0\left[\sum_{t=0}^{\infty}\alpha^t\left|g(i_t,i_{t+1})\right|\right]<\infty .$$

Because $\pi(i) > 0$ for all i, the expectation defining $J(i)$ is well-defined and finite.

Using Fubini's Theorem to switch the order of expectation and summation in the definition of J, we obtain

$$\begin{aligned} J(i) &:= E\left[\sum_{t=0}^{\infty}\alpha^t g(i_t,i_{t+1})\middle|i_0=i\right] \\ &= \sum_{t=0}^{\infty}\alpha^t E\left[g(i_t,i_{t+1})\middle|i_0=i\right] \\ &= \sum_{t=0}^{\infty}\alpha^t E\left[\bar{g}(i_t)\middle|i_0=i\right], \end{aligned}$$

and it follows that

$$J=\sum_{t=0}^{\infty}(\alpha P)^t\,\bar{g}\,.$$

To show that J is in $L_2(X, D)$, we have

$$\begin{aligned} \|J\|_D &\le \sum_{t=0}^{\infty}\left\|(\alpha P)^t\,\bar{g}\right\|_D \\ &\le \sum_{t=0}^{\infty}\alpha^t\,\|\bar{g}\|_D \\ &= \frac{\|\bar{g}\|_D}{1-\alpha}, \end{aligned}$$

where the second inequality follows from Lemma 3.A.1. Furthermore, by Assumption 3.3 (b), we have

$$\begin{aligned} \|\bar{g}\|_D^2 &= \sum_{i\in X}\pi(i)\left(\sum_{j\in X}p_{ij}g(i,j)\right)^2 \\ &\le \sum_{i\in X}\pi(i)\sum_{j\in X}p_{ij}g^2(i,j) \\ &= E_0\left[g^2(i_t,i_{t+1})\middle|i_t=i\right] \\ &< \infty. \end{aligned}$$

Therefore, J is in $L_2(X, D)$. **q.e.d.**

Lemma 3.A.3: *Under Assumption 3-3, for any* $J\in B(S,D)$ *and* $\lambda\in[0,1]$, $T^{(\lambda)}J$ *is in* $L_2(X, D)$*, and we have*

$$T^{(\lambda)}J = (1-\lambda)\sum_{m=0}^{\infty}\lambda^m\left(\sum_{t=0}^{m}(\alpha P)^t\,\overline{g} + (\alpha P)^{m+1}J\right).$$

Proof: We have

$$\begin{aligned}\left(T^{(\lambda)}J\right)(i) &= (1-\lambda)\sum_{m=0}^{\infty}\lambda^m E\left[\sum_{t=0}^{m}a^t g_t(i_t, i_{t+1}) + \alpha^{m+1}J(i_{m+1}) \mid i_0 = i\right]\\ &= (1-\lambda)\sum_{m=0}^{\infty}\lambda^m\left(\sum_{t=0}^{m}a^t E[\overline{g}(i_t)|i_0 = i] + \alpha^{m+1}E[J(i_{m+1}) \mid i_0 = i]\right).\end{aligned}$$

Because in Lemma 3.A.2 we have shown that $\|\overline{g}\|_D^2 < \infty$, then, for $\lambda < 1$, we use Lemma 3.A.1 to obtain

$$\left\|(1-\lambda)\sum_{m=0}^{\infty}\lambda^m\sum_{t=0}^{m}(\alpha P)^t\,\overline{g}\right\|_D \le (1-\lambda)\sum_{m=0}^{\infty}\lambda^m\sum_{t=0}^{m}\alpha^t\|\overline{g}\|_D < \infty.$$

Similarly,

$$\begin{aligned}\left\|(1-\lambda)\sum_{m=0}^{\infty}\lambda^m(\alpha P)^{m+1}J\right\|_D &\le (1-\lambda)\sum_{m=0}^{\infty}\lambda^m\alpha^{m+1}\|J\|_D\\ &\le \alpha\|J\|_D,\end{aligned}$$

for any $J \in L_2(X, D)$. **q.e.d.**

Lemma 3.A.4 *Under Assumption 3-3(a), for any* $\alpha \in [0,1)$, $J, \overline{J} \in L_2(S, D)$ *and* $\lambda \in [0,1]$, *we have*

$$\left\|T^{(\lambda)}J - T^{(\lambda)}\overline{J}\right\|_D \le \frac{\alpha(1-\lambda)}{1-\alpha\lambda}\left\|J-\overline{J}\right\|_D \le \alpha\left\|J-\overline{J}\right\|_D.$$

Proof: The case of $\lambda = 1$ is trivial. For $\lambda < 1$, by Lemma 3.A.1 and 3.A.3, we have

$$\begin{aligned}\left\|T^{(\lambda)}J - T^{(\lambda)}\overline{J}\right\|_D &= \left\|(1-\lambda)\sum_{m=0}^{\infty}\lambda^m(\alpha P)^{m+1}\left(J-\overline{J}\right)\right\|_D\\ &\le (1-\lambda)\sum_{m=0}^{\infty}\lambda^m\alpha^{m+1}\left\|J-\overline{J}\right\|_D\\ &= \frac{\alpha(1-\lambda)}{1-\alpha\lambda}\left\|J-\overline{J}\right\|_D\\ &\le \alpha\left\|J-\overline{J}\right\|_D.\end{aligned}$$

q.e.d.

Lemma 3.A.5 *Under Assumption 3-3, for any* $\lambda \in [0,1]$, *the exact J function uniquely solves the system of equations given by* $J = T^{(\lambda)} J$.

Proof: If $\lambda = 1$, the result follows directly from the definition of operator $T^{(\lambda)}$. For $\lambda \in [0,1)$, by Lemma 3.A.2 and 3.A.3, using Fubini's theorem, we have

$$\begin{aligned}
T^{(\lambda)} J &= (1-\lambda) \sum_{m=0}^{\infty} \lambda^m \left(\sum_{t=0}^{m} (\alpha P)^t \, \bar{g} + (\alpha P)^{m+1} J \right) \\
&= (1-\lambda) \sum_{m=0}^{\infty} \lambda^m \left(\sum_{t=0}^{m} (\alpha P)^t \, \bar{g} + (\alpha P)^{m+1} \sum_{t=0}^{\infty} (\alpha P)^t \, \bar{g} \right) \\
&= (1-\lambda) \sum_{m=0}^{\infty} \lambda^m \left(\sum_{t=0}^{\infty} (\alpha P)^t \, \bar{g} \right) \\
&= (1-\lambda) \sum_{m=0}^{\infty} \lambda^m J.
\end{aligned}$$

The contraction property by Lemma 3.A.4 implies that J is the unique fixed point of $T^{(\lambda)}$.

q.e.d.

Lemma 3.A.6 *Under Assumptions 3-3, and 3-4,* $\prod T^{(\lambda)}(\cdot)$ *is a contraction and has a unique fixed point which is of the form* Φr^* *for a unique choice of* r^*. *Furthermore,* r^* *satisfies the following bound:*

$$\left\| \Phi r^* - J \right\|_D \le \frac{(1-\lambda\alpha)}{1-\alpha} \left\| \prod J - J \right\|_D .$$

Proof: Lemma 3.A.4 ensures that $T^{(\lambda)}$ is a contraction on $L_2(X, D)$, and Lemma 3.A.5 ensures that J is the fixed point of the contraction. Note that for $J \in L_2(X, D)$ we have

$$\|J\|_D^2 = \left\|\prod J\right\|_D^2 + \left\|J - \prod J\right\|_D^2,$$

since $\prod J \perp_D \left(J - \prod J\right)$. It follows that projection $\prod$ is non-expansive, and thus the composition $\prod T^{(\lambda)}(\cdot)$ is a contraction. Therefore, $\prod T^{(\lambda)}(\cdot)$ has a unique fixed point of the form Φr^*, for some r^*. Because basis functions $\phi_j(\cdot)$ are assumed linearly independent, it follows that the choice of r^* is unique.

Using the fact that J is in $L_2(X, D)$ by Lemma 3.A.2 and is the fixed point of $T^{(\lambda)}$ by Lemma 3.A.5, we establish the desired bound. In particular, we have

$$\begin{aligned}\left\|\Phi r^* - J\right\|_D &\le \left\|\Phi r^* - \Pi J\right\|_D + \left\|\Pi J - J\right\|_D \\ &= \left\|\Pi T^{(\lambda)}\left(\Phi r^*\right) - \Pi J\right\|_D + \left\|\Pi J - J\right\|_D \\ &\le \left\|T^{(\lambda)}\left(\Phi r^*\right) - J\right\|_D + \left\|\Pi J - J\right\|_D \\ &\le \frac{\alpha(1-\lambda)}{1-\alpha\lambda}\left\|\Phi r^* - J\right\|_D + \left\|\Pi J - J\right\|_D,\end{aligned}$$

and it follows that

$$\left\|\Phi r^* - J\right\|_D \le \frac{\left\|\Pi J - J\right\|_D}{1-\alpha(1-\lambda)/(1-\alpha\lambda)} = \frac{1-\alpha\lambda}{1-\alpha}\left\|\Pi J - J\right\|_D.$$

q.e.d.

The following lemmas are to characterize the expected behaviour of the steps taken by the TD(λ) algorithm in "steady-state." A convenient representation of TD(λ) can be obtained by defining a sequence of *eligibility vectors* z_t by

$$\begin{aligned} z_t &= \sum_{k=0}^{t}(\alpha\lambda)^{t-k}\nabla_r\tilde{J}(i_k, r_k) \\ &= \sum_{k=0}^{t}(\alpha\lambda)^{t-k}\phi(i_k).\end{aligned}$$

Using the eligibility vector z we further construct a process $Y_t = (i_t, i_{t+1}, z_t)$. Since i_t is a Markov process, it is easy to see that Y_t is a Markov process too. At each time t, the random vector Y_t, together with the current parameter vector r_t, provides all necessary information for computing r_t. In this regard, we define a function s with

$$s(r,Y) = \left[g(i,j) + \alpha\tilde{J}(j,r) - \tilde{J}(i,r)\right]z, \quad \forall i,j \in X,$$

where $Y = (i, j, z)$. Thus, we can rewrite the TD(λ) algorithm as

$$r_{t+1} = r_t + \eta_t s(r_t, Y_t).$$

Of particular interest to us is the behaviour of s in steady state, i.e. $E_0[s(r,Y_t)]$ for any given r, as s determines the correction to r. Prior to studying $E_0[s(r,Y_t)]$, we establish a few preliminary relations in the next lemma.

Lemma 3.A.7 *Under Assumption 3-3, and 3-4, the following relations hold:*

(a) $E_0\left[\phi(i_t)\phi'(i_{t+m})\right] = \Phi' D P^m \Phi,$ *for* $m \geq 0,$

(b) there exists a finite constant G such that $\left\| E_0\left[\phi(i_t)\phi'(i_{t+m})\right]\right\| \leq G$*, for all* $m \geq 0,$

(c) $E_0\left[z_t\phi'(i_t)\right] = \sum_{m=0}^{\infty} (\alpha\lambda)^m \Phi' D P^m \Phi,$

(d) $E_0\left[z_t\phi'(i_{t+1})\right] = \sum_{m=0}^{\infty} (\alpha\lambda)^m \Phi' D P^{m+1} \Phi,$

(e) $E_0\left[z_t g(i_t, i_{t+1})\right] = \sum_{m=0}^{\infty} (\alpha\lambda)^m \Phi' D P^{m+1} \bar{g}.$

Further more, each of the above expressions is well defined and finite.

Proof: For any $J, \bar{J} \in L_2(X, D)$, we have

$$\begin{aligned} E_0\left[J(i_t)\bar{J}(i_{t+m})\right] &= \sum_{i\in X}\pi(i)\sum_{j\in X}\Pr\left(i_{t+m}=j \mid i_t = i\right)J(i)\bar{J}(j) \\ &= \sum_{i\in X}\pi(i)J(i)\left[P^m\bar{J}\right](i) \\ &= J' D P^m \bar{J}. \end{aligned}$$

By Lemma 3.A.1 and using the Cauchy-Schwarz inequality, $J' D P^m \bar{J}$ is finite. Because $J = \Phi r$ and $\bar{J} = \Phi\bar{r}$, we obtain

$$E_0\left[r'\phi(i_t)\phi'(i_{t+m})\bar{r}\right] = r'\Phi' D P^m \Phi \bar{r},$$

which is equivalent to

$$E_0\left[\phi(i_t)\phi'(i_{t+m})\right] = \Phi' D P^m \Phi.$$

We then place a bound on the Euclidean norm $\| \Phi' D P^m \Phi \|$ as follows:

$$\begin{aligned} \left\|\Phi' D P^m \Phi\right\| &\leq K^2 \max_{k,j}\left|\phi_k{}' D P^m \phi_j\right| \\ &= K^2 \max_{k,j}\left|\phi_k' D^{\frac{1}{2}} D^{\frac{1}{2}} P^m \phi_j\right| \\ &\leq K^2 \max_{k,j}\left\|\phi_k'\right\|_D \left\|P^m\phi_j\right\|_D \\ &\leq K^2 \max_{k}\left\|\phi_k\right\|_D^2 \\ &= K^2 \max_{k} E_0\left[\phi_k^2(i)\right]. \end{aligned}$$

The result $K^2 \max_k E_0\left[\phi_k^2(i)\right]$ is a finite constant *G*, by Assumption 3-4(b). This far, we have verified parts (a) and (b) of Lemma 3.A.7.

We now begin with the analysis for part (c). Because $E_0[z_0\phi'(i_t)]$ is the same for all *t*, it suffices to prove the result for the case *t* = 0. We have

$$E_0\left[z_0\phi'(i_0)\right] = E_0\left[\sum_{\tau=-\infty}^{0}(\alpha\lambda)^{-\tau}\phi(i_\tau)\phi'(i_0)\right]$$
$$= \sum_{\tau=-\infty}^{0}(\alpha\lambda)^{-\tau}E_0\left[\phi(i_\tau)\phi'(i_0)\right],$$

where the interchange of summation and expectation is justified by the dominated convergence theorem. By part (a), we establish part (c) with the result. The proofs for (d) and (e) contain entirely similar arguments, and hence omitted. **q.e.d.**

The next lemma characterises $E_0[s(r, Y(t))]$.

Lemma 3.A.8 *Under Assumption 3-3 and 3-4, we have*

$$E_0\left[s(r,Y_t)\right] = \Phi' D\left(T^{(\lambda)}(\Phi r) - \Phi r\right),$$

which is well defined and finite for any finite r.

Proof: By applying Lemma 3.A.7, we have

$$E_0\left[s(r,Y_t)\right] = E_0\left[z(t)g(i_t,i_{t+1}) + \alpha z(t)\phi'(i_{t+1})r - z(t)\phi'(i_t)r\right]$$
$$= \Phi' D\sum_{m=0}^{\infty}(\alpha\lambda P)^m\left(\bar{g} + \alpha P\Phi r - \Phi r\right).$$

For λ = 1, it follows that

$$E_0\left[s(r,Y_t)\right] = \Phi' D\left(J - \Phi r\right).$$

For $\lambda \in [0,1)$ and any $J \in L_2(X,D)$, we have

$$\sum_{m=0}^{\infty}(\alpha\lambda P)^m J = (1-\lambda)\sum_{m=0}^{\infty}\lambda^m\sum_{t=0}^{\infty}(\alpha P)^t J.$$

and therefore by Lemma 3.A.3, we have

$$E_0\left[s(r,Y_t)\right] = \Phi' D\left((1-\lambda)\sum_{m=0}^{\infty}\lambda^m\sum_{t=0}^{\infty}(\alpha P)^t\bar{g} + \left((1-\lambda)\sum_{m=0}^{\infty}\lambda^m(\alpha P)^{m+1} - I\right)\Phi r\right)$$
$$= \Phi' D\left(T^{(\lambda)}(\Phi r) - \Phi r\right).$$

Each expression is finite and well defined by Lemma 3.A.7. **q.e.d.**

The next lemma shows that the steps taken by TD(λ) tend to move r_t towards r^*.

Lemma 3.A.9 *Under Assumption 3-3 and 3-4, we have*

$$\left(r-r^*\right)' E_0\left[s\left(r,Y_t\right)\right]<0, \quad \forall r \neq r^*.$$

Proof: Using Lemma 3.A.8 we have

$$\begin{aligned}\left(r-r^*\right)' \Phi' D\left(T^{(\lambda)}\left(\Phi r\right)-\Phi r\right) &= \left(r-r^*\right)' \Phi' D\left(\left(I-\prod\right)T^{(\lambda)}\left(\Phi r\right)+\prod T^{(\lambda)}\left(\Phi r\right)-\Phi r\right)\\ &= \left(\Phi r-\Phi r^*\right)' D\left(\prod T^{(\lambda)}\left(\Phi r\right)-\Phi r\right),\end{aligned}$$

where the last equality follows because $\Phi' D\prod = \Phi' D$, since projection $\prod$ can be expressed as

$$\prod = \Phi'\left(\Phi' D\Phi\right)^{-1}\Phi' D.$$

As shown in the proof of Lemma 3.A.5, $\prod T^{(\lambda)}$ is a contraction with fixed point Φr^*, and the contraction factor is no larger than α. Hence,

$$\left\|\prod T^{(\lambda)}\left(\Phi r\right)-\Phi r^*\right\|_D \leq \alpha\left\|\Phi r-\Phi r^*\right\|_D,$$

and using the Cauchy-Schwartz inequality, we have

$$\begin{aligned}\left(r-r^*\right)' \Phi' D\left(T^{(\lambda)}\left(\Phi r\right)-\Phi r\right) &= \left(\Phi r-\Phi r^*\right)' D\left(\prod T^{(\lambda)}\left(\Phi r\right)-\Phi r^*+\left(\Phi r^*-\Phi r\right)\right)\\ &\leq \left\|\Phi r-\Phi r^*\right\|_D \cdot \left\|\prod T^{(\lambda)}\left(\Phi r\right)-\Phi r^*\right\|_D + \left\|\Phi r^*-\Phi r\right\|_D^2\\ &\leq \left(\alpha-1\right)\left\|\Phi r-\Phi r^*\right\|_D^2.\end{aligned}$$

Since $\alpha < 1$, the result follows. **q.e.d.**

We now state without proof a result concerning stochastic approximation, which will be used in the proof of Theorem 3.2. This is a special case of very general result on stochastic approximation algorithm (Theorem 17, pp. 239, Benveniste *et al.*, 1990).

Theorem 3.A.1 *Consider an iterative algorithm of the form*

$$r_{t+1} = r_t + \eta_t\left(A\left(Y_t\right)r_t + c\left(Y_t\right)\right)$$

where:

(a) The learning rate, (or stepsize), η_t satisfies Assumption 3-6(a).

(b) Y_t is a Markov process with a steady-state distribution, and there exists a mapping h from the states of the Markov process to the positive real numbers, satisfying the remaining conditions. Let $E_0[\cdot]$ stands for expectation with respect to this steady-state distribution.

(c) $A(\cdot)$ and $c(\cdot)$ are matrix and vector valued functions in respect, for which $A= E_0[A(Y_t)]$ and $c = E_0[c(Y_t)]$ are well defined and finite.

(d) The matrix A is negative definite.

(e) There exist constants C and β such that, for all Y,

$$\sum_{t=0}^{\infty}\left\| E\left[A\left(Y_t\right)\middle|Y_0=Y\right]-A\right\| \le C\left(1+h^{\beta}\left(Y\right)\right),$$

and

$$\sum_{t=0}^{\infty}\left\| E\left[c\left(Y_t\right)\middle|Y_0=Y\right]-c\right\| \le C\left(1+h^{\beta}\left(Y\right)\right).$$

(f) For any $\beta > 1$ there exists a constant θ_β such that for all Y, t,

$$E\left[h^{\beta}\left(Y_t\right)\middle|Y_0=Y\right] \le \theta_\beta\left(1+h^{\beta}\left(Y\right)\right).$$

Then, r_t converges to r^*, with probability 1, where r^* is the unique vector that satisfies

$$Ar^* + c = 0.$$

Finally we come to the proof of Theorem 3.2.

Proof of Theorem 3.2:

Function $s(r_t, Y_t)$ for updating r_t is

$$r_{t+1} = r_t + \eta_t s\left(r_t, Y_t\right)$$

where

$$\begin{aligned} s\left(r_t,Y_t\right) &= z_t g\left(i_t,i_{t+1}\right) + z_t\left(\alpha\tilde{J}\left(i_{t+1},r_t\right) - \tilde{J}\left(i_t,r_t\right)\right) \\ &= z_t g\left(i_t,i_{t+1}\right) + z_t\left(\alpha\phi'\left(i_{t+1}\right) - \phi'\left(i_t\right)\right)r_t. \end{aligned}$$

This function can be rewritten with A and c in the form

$$s\left(r_t,Y_t\right) = A\left(Y_t\right)r_t + c\left(Y_t\right).$$

where

$$A\left(Y_t\right) = z_t\left(\alpha\phi'\left(i_{t+1}\right) - \phi'\left(i_t\right)\right),$$

and

$$c(Y_t) = z_t g(i_t, i_{t+1}).$$

By Lemma 3.A.7, we have

$$\begin{aligned} A &:= E_0\left[A(Y_t)\right] \\ &= E_0\left[z_t\left(\alpha\phi'(i_{t+1}) - \phi'(i_t)\right)\right] \\ &= \sum_{m=0}^{\infty}(\alpha\lambda)^m \Phi' D P^{m+1}\Phi - \sum_{m=0}^{\infty}(\alpha\lambda)^m \Phi' D P^m \Phi \\ &= \Phi' D(P - I)\sum_{m=0}^{\infty}(\alpha\lambda P)^m \Phi, \end{aligned}$$

and similarly

$$c := \Phi' D \sum_{m=0}^{\infty}(\alpha\lambda P)^m \bar{g}.$$

Matrix A and vector c are both well defined and finite.

By Lemma 3.A.6, we have $\prod T^{(\lambda)}(\Phi r^*) = \Phi r^*$. From Lemma 3.A.9, we have $\Phi' D \prod = \Phi' D$, and therefore, $\Phi' D\, T^{(\lambda)}(\Phi r^*) = \Phi' D\, \Phi r^*$. Using the formula of $E_0[s(r^*, Y_t)]$, as given by Lemma 3.A.8, we can conclude that $E_0[s(r^*, Y_t)] = 0$. Hence,

$$\begin{aligned} A(r - r^*) &= E_0\left[s(r, Y_t)\right] - E_0\left[s(r^*, Y_t)\right] \\ &= E_0\left[s(r, Y_t)\right]. \end{aligned}$$

It follows from Lemma 3.A.9 that

$$(r - r^*)' A (r - r^*) < 0, \quad \forall r \neq r^*,$$

and thus A is negative definite.

Theorem 3.A.1 is used here to show that r_t converges. The analysis so far ensures that all conditions except for (e) and (f) are met. In the following analysis, we show that Assumption 3-5 is sufficient to ensure validity for these two conditions.

We begin by binding the summations involved in condition (e). Using the formula for z_t, we have

$$E\left[z_t\phi'(i_t)\middle|Y_0\right]-E_0\left[z_t\phi'(i_t)\right]=(\alpha\lambda)^{t+1}E\left[z_{(-1)}\phi(i_t)\right]$$
$$+\sum_{m=0}^{t}(\alpha\lambda)^m E\left[\phi(i_{t-m})\phi'(i_t)\middle|Y_0\right]$$
$$-\sum_{m=0}^{\infty}(\alpha\lambda)^m E_0\left[\phi(i_{t-m})\phi'(i_t)\right].$$

Using the triangle inequality, we obtain

$$\sum_{t=0}^{\infty}\left\|E\left[z_t\phi'(i_t)\middle|Y_0\right]-E_0\left[z_t\phi'(i_t)\right]\right\|\le\sum_{t=0}^{\infty}(\alpha\lambda)^{t+1}\left\|E\left[z_{-1}\phi'(i_t)\middle|Y_0\right]\right\|$$
$$+\sum_{t=0}^{\infty}\sum_{m=0}^{t}(\alpha\lambda)^m\left\|E\left[\phi(i_{t-m})\phi'(i_t)\middle|Y_0\right]-E_0\left[\phi(i_{t-m})\phi'(i_t)\right]\right\|$$
$$+\sum_{t=0}^{\infty}\sum_{m=t+1}^{\infty}(\alpha\lambda)^m\left\|E_0\left[\phi(i_{t-m})\phi'(i_t)\right]\right\|.$$

We will individually bind the magnitude of each summation in the right hand side. First, we have

$$\sum_{t=0}^{\infty}(\alpha\lambda)^{t+1}\left\|E\left[z_{-1}\phi'(i_t)\middle|Y_0\right]\right\|=\left\|z_{-1}\right\|\sum_{t=0}^{\infty}(\alpha\lambda)^{t+1}\left\|E\left[\phi(i_t)\middle|Y_0\right]\right\|$$
$$=\frac{1}{\alpha\lambda}\left\|z_0-\phi(i_0)\right\|\sum_{t=0}^{\infty}(\alpha\lambda)^{t+1}\left\|E\left[\phi(i_t)\middle|Y_0\right]\right\|,$$

where the second inequality follows from the fact that

$$z_0=(\alpha\lambda)z_{-1}+\phi(i_0).$$

Assumption 3-5(a) implies that

$$\left\|E\left[\phi(i_t)\middle|Y_0\right]\right\|\le C\left(1+f(i_0)+f(i_1)\right)^{\beta},$$

for some constant C and β and any $t\ge 0$. It follows that

$$\sum_{t=0}^{\infty}(\alpha\lambda)^{t+1}\left\|E\left[z_{-1}\phi'(i_t)\middle|Y_0\right]\right\|\le C\left(1+\|z_0\|+f(i_0)+f(i_1)\right)^{\beta}.$$

Next, we deal with the second summation. Letting $\Delta(t\text{-}m,\ t)$ be defined as

$$\Delta(t-m,t)=\left\|E\left[\phi(i_{t-m})\phi'(i_t)\middle|Y_0\right]-E_0\left[\phi(i_{t-m})\phi'(i_t)\right]\right\|,$$

we have

$$\sum_{t=0}^{\infty}\sum_{m=0}^{t}(\alpha\lambda)^m\Delta(t-m,t)=\sum_{m=0}^{\infty}(\alpha\lambda)^m\left(\Delta(0,m)+\sum_{t=m+1}^{\infty}\Delta(t-m,t)\right)$$
$$\le C\left(f(i_0)+f(i_1)\right),$$

where the inequality follows from Assumption 3-5(a).

Finally, from Lemma 3.A.7, we know that $E_0\left[\phi(i_{t-m})\phi'(i_t)\right] \le G$, for some constant *G*. And we have

$$\begin{aligned}\sum_{t=0}^{\infty}\sum_{m=t+1}^{\infty}(\alpha\lambda)^m\left\|E_0\left[\phi(i_{t-m})\phi'(i_t)\right]\right\| &\le G\sum_{t=0}^{\infty}\sum_{m=t+1}^{\infty}(\alpha\lambda)^m\\ &= G\sum_{t=0}^{\infty}\frac{(\alpha\lambda)^{t+1}}{1-\alpha\lambda}\\ &< \infty.\end{aligned}$$

Given these bounds, it follows that there exist positive constants *C* and *β* such that

$$\sum_{t=0}^{\infty}\left\|E\left[z_t\phi'(i_t)|Y_0\right]-E_0\left[z_t\phi'(i_t)\right]\right\| \le C\left(1+\|z_0\|+f(i_0)+f(i_1)\right)^{\beta}.$$

It can be seen that the summation above is bounded by a polynomial function on the right hand side. An identical argument can be produced for the terms $\alpha z_t\phi'(i_{t+1})$, and $z_t g(i_t, i_{t+1})$, which is omitted here. Using these arguments, we place bounds that are polynomial in $\|z_0\|$, $f(i_0)$, and $f(i_1)$, on the summation in Condition (e) of Theorem 3.A.1. We can thus satisfy the condition with a function $h(Y)$ that is polynomial in $\|z_0\|$, $f(i_0)$, and $f(i_1)$. Following Assumption 3-5(b), such a function $h(Y)$ satisfies Condition (f) in Theorem 3.A.1.

We now have all the conditions satisfied to apply Theorem 3.A.1. It follows that r_t converges to r^*, which solves $Ar^*+c = 0$. Since $Ar^*+c = E_0[s(r^*, Y_t)]$, Lemma 3.A.8 implies that

$$\Phi' D\left(T^{(\lambda)}\left(\Phi r^*\right)-\Phi r^*\right)=0.$$

Using Lemma 3.A.6, and under Assumption 3-5, r^* uniquely satisfies this equation and is the unique fixed point of $\prod T^{(\lambda)}$, with a bounded error. This completes the proof of Theorem 3.2.

CHAPTER 4 ADP ALGORITHMS FOR ADAPTIVE TRAFFIC SIGNAL CONTROL

In the preceding chapter, we introduced the fundamentals of approximate dynamic programming (ADP) for real-time control. In this chapter we present the development of ADP algorithm for adaptive traffic signal operation.

We begin discussion with a few general assumptions of traffic signal operation in Section 4.1. The general assumptions reflect the conditions common to traffic engineering. The methods for generating random traffic in computer simulation are discussed in Section 4.2. State definition and dynamics of state transition are discussed in Section 4.3. Based on the formalism of state transition, we introduce dynamic programming (DP) formulae in Section 4.4. Structural properties of the value function in DP are discussed in Section 4.5. The structural properties of J function are helpful in determining the structure of approximation functions, which is discussed in Section 4.6. General control policy is discussed in Section 4.7, and traffic models in Section 4.8. The ADP algorithms are shown in Section 4.9. A summary is provided in Section 4.10.

4.1 General Assumptions

The general assumptions frame the scope of this study so that the features key to our interest are highlighted, and those less significant simplified or neglected.

The first two assumptions concern traffic signals and their indication durations:

Assumption 4-1 *Signal phases are represented by effective greens and effective reds only, thus excluding the effects of amber intervals.*

Assumption 4-2 *There are no constraints on the maximum duration of a green period. A signal switch is immediately followed by inter-green and minimum green.*

We consider discrete time controllers only in this study. To investigate the impact of different discrete temporal *resolutions*, we consider three cases: 5s, 2s and 0.5s per temporal increment. The following two assumptions apply to all of the three resolutions.

Assumption 4-3 *Queue lengths are calculated at the end of each temporal interval, neglecting the detail of vehicle behaviour during the interval. Signals may only be switched at the boundary between intervals.*

Assumption 4-4 *The saturation flow on all lanes is 2 vehicles per 5 seconds.*

This rate is equivalent to 1440 vehicles per hour, which is close to the saturation flow rate of a single traffic lane.

The next assumption concerns about *lost time* in traffic control. The term 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 be assumed to be 2.0s (Roes *et al.*, 2004). In this study, lost time is not modelled for the purpose of preserving simplicity.

Assumption 4-5 *There is no lost time for vehicles receiving green signal.*

To reflect the norm of real-time traffic signal control, we consider traffic sensors that detect incoming vehicles. We further assume that

Assumption 4-6 *Upstream roadside sensors provide information of arriving traffic of the next 10 seconds.*

4.2 Traffic Generation

This section discusses random traffic generation in computer simulation. At 5s resolution, traffic arrivals during each time interval take integer value of 0, 1, or 2 vehicles only; at 2s and 0.5s resolutions, traffic arrival per temporal interval is a binary variable taking the value of either 0 or 1.

Random traffic arrivals at 5s and 2s resolutions are generated by binomial distribution. The maximum rate of 2 vehicles per time interval is set for the 5s resolution so that the arrival rate could not exceed the capacity. Letting Q be the arriving rate, we denote the cumulative distribution function by

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

and probability of $q = m$, is

$$P\{q=m\} = \binom{n}{m} p^m (1-p)^{n-m}, \; m = 0,1,\ldots,n.$$

where $n = 2$ at 5s resolution, and $n = 1$ at 2s resolution.

Simulation of the random arrival process is then generated by using *Inverse Transformation Method* (Devroye, 1986), which first generates a *uniform random number z* between 0 and 1, and then sets *F(Q)=z* and solve for *Q* to produce the desired random observation from the probability distribution.

In order to generate traffic at 0.5s resolution, a shifted Bernoulli process is used to ensure that an appropriate minimum inter-arrival time is respected. With probability P, an arrival is generated and the trial has duration $T_p=n_b\Delta t$; with probability (1-P), no arrival is generated and the trial has duration $T_p=\Delta t$. The mean number of vehicles generated in a single trial is E(N) = P, and the mean duration of the trial is E(T_p) = [Pn_b+ (1-P)] Δt. The mean rate of traffic generation is given by:

$$q = \frac{E\{N\}}{E\{T_p\}} = \frac{P}{\left[1+P\left(n_b-1\right)\right]\Delta t}. \tag{4-1}$$

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

$$P = \frac{Q\Delta t}{1-(n_b-1)Q\Delta t}. \tag{4-2}$$

The rest of the process for generating random traffic at the 0.5s resolution is identical to that of the other two resolutions. Because Assumption 4-3 stipulates that queue lengths are calculated at the end of each interval, vehicle delays measured under the different resolution are not directly comparable. In particular, in the case of 5s resolution, no delay is attributed to either vehicle in a time increment during which two depart, whilst under 2s and 0.5s resolutions, one of the vehicles will be delayed for the whole of the departure time of the other. To facilitate performance comparison at different resolutions, we record the decision series of the coarse case and implement them in the fine case to calculate delay. The performance measures thus obtained are then comparable. Specifications for generating traffic data at different resolutions are summarised in Table 4-1.

Table 4-1 Random Traffic Generation by ITM

Resolution (in seconds)	Traffic Rate	Random Process
5.0	0, 1, or 2	Binomial
2.0	0 or 1	Binomial
0.5	0 or 1	Shifted Bernoulli

4.3 State Transition

The state of traffic intersection has two primary components: the queue state of individual traffic link and the controller state that describes the signal indication to individual link. We denote the queue state by the column vector *l* and controller state by the column vector *s*. For an intersection of *N* traffic links, the vectors *l* and *s* can be expressed as:

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

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

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

We denote decision vector *u* as:

$$u = \begin{bmatrix} u(1) \\ \vdots \\ u(N) \end{bmatrix},$$

where the decision variable takes

$$u_t(n) = \begin{cases} 1 \text{ if signal } n \text{ is switched at time } t \\ 0 \text{ unchanged.} \end{cases} \tag{4-3}$$

In particular, Assumption 4-3 requires that traffic signal may only be switched at the boundary between intervals. From this point, we further assume that

Assumption 4-7 *For any $u(n) \in \{0,1\}$, $n = 1, 2, \ldots, N$, decision $u(n)$ takes effects at the beginning of the temporal increment from t to t+1.*

To facilitate further discussion on system dynamics, we introduce the post-decision vector $s_{u,t}$ where

$$s_{u,t} = \begin{bmatrix} s_{u,t}(1) \\ \vdots \\ s_{u,t}(N) \end{bmatrix},$$

which describes the state of signals after implementing decision u_t. The transition from s_t to $s_{u,t}$ can be expressed as

$$s_{u,t}(n) = \left(s_t(n) + u_t(n)\right)\text{mod}_2, \tag{4-4}$$

At fine resolutions (2.0s and 0.5s), state *i* includes additional dimensions. This is because that minimum green and inter-green may be greater than temporal increment at the fine resolutions. Minimum green and inter-green are mandatory intervals during which no signal switch is admissible. To enforce the mandatory regulations, we introduce a constraint variable m_c. Let Δt denote temporal increment (in seconds), T_{min} the minimum green, T_{inter} the inter-green, and M_c the total number of temporal intervals that correspond to mandatory inter-green and minimum green, we have the following relationship

$$M_c = \left(T_{\min} + T_{\text{inter}}\right)/\Delta t - 1\,. \tag{4-5}$$

When signal switch is made, we set constraint variable:

$$m_c = M_c\,. \tag{4-6}$$

For each time increment after the signal switch, the constraint variable is recursively calculated as

$$m_c = \max\left\{0, m_c - 1\right\}. \tag{4-7}$$

Signal switch is admissible only at times when $m_c = 0$. In this way, we guarantee that mandatory timings are enforced, i.e. successive signal switches only occur after inter-green and minimum green periods have been satisfied.

At the 0.5s resolution, in accordance to the traffic generation rules specified in Eq. (4-1) and (4-2), we introduce vector $m_d(n)$ for each traffic link to deterministically ensure that no more than one departure within the duration $(M_d + 1)\Delta t$, where in our case

$$M_d = Ceil\left[\left(y^*\Delta t\right)^{-1}\right], \tag{4-8}$$

where function $Ceil\,(x)$ returns the least integer that is greater than x, and y^* is the saturation departure rate. This relationship guarantees that saturation departure rate is 2 vehicles per 5.0s.

Upon each vehicle departure

$$m_d\left(n\right) = M_d\,. \tag{4-9}$$

For each time increment after the precedent departure, we have

$$m_d\left(n\right) = \max\left\{0, m_d\left(n\right) - 1\right\}. \tag{4-10}$$

where the column vector m_d can be expressed as

$$m_d = \begin{bmatrix} m_d\left(1\right) \\ \vdots \\ m_d\left(N\right) \end{bmatrix}$$

for an intersection of total N traffic links.

A general form of state i for all resolutions can be written as

$$i = \left[l, s, m_c, m_d\right], \tag{4-11}$$

where $i\,(n) = [\,l\,(n), s\,(n), m_c, m_d(n)\,]$.

Random arriving traffic detected by the roadside sensors is denoted by column vector w, where

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

and

$$w(n) \in \begin{cases} \{0,1,2\} & \text{5s resolution} \\ \{0,1\} & \text{2s and 0.5s resolution.} \end{cases} \tag{4-12}$$

The departing traffic is denoted by column vector *y*. For the *N*-link intersection, vector *y* can be expressed as

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

where at 5s resolution, we have

$$y(n) = \begin{cases} \min\{y^*, l(n) + w(n)\} & \text{if } s_u(n) = 0 \\ 0 & \text{if } s_u(n) = 1, \end{cases} \tag{4-13}$$

and at 2s and 0.5s resolutions, we have

$$y(n) = \begin{cases} \min\{y^*, l(n) + w(n)\} & \text{if } s_u(n) = 0 \text{ and } m_d(n) = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4-14}$$

Finally, the transition of system state during time increment *t* obeys the following rules: Signal vector *s*,

$$s_{t+1}(n) = s_{u,t}(n). \tag{4-15}$$

where $s_{u,t}$ is transformed by Eq. (4-4). For the queue vector *l*, we have

$$l_{t+1}(n) = l_t(n) - y_t(n) + w_t(n), \tag{4-16}$$

Overall, we can define a transition function *f* (·) such that

$$f\left(l_t(n), s_t(n), m_c, m_d(n), w_t(n), u_t(n)\right) \equiv l_t(n) - y_t(n), \tag{4-17}$$

and rewrite Eq. (4-16) as

$$l_{t+1}(n) = f\left(l_t(n), s_t(n), m_c, m_d(n), w_t(n), u_t(n)\right) + w_t(n), \tag{4-18}$$

which, by using vector forms, can be further simplified as

$$l_{t+1} = f\left(i_t, w_t, u_t\right) + w_t. \tag{4-19}$$

We now investigate into the structural properties of the value function in dynamic programming.

4.4 Dynamic Programming Formulae

Given the definitions of state variables in Section 4.3, we define the one-step cost function for an isolated intersection of *N* links as:

$$g(i,w,u)=\sum_{n=1}^{N}\left[l(n)-y(n)+w(n)\right]\Delta t\,. \tag{4-20}$$

This means that the one-step cost is the sum of vehicle-seconds occurred during a temporal interval of Δt seconds. Recalling equations (3-1) and (3-2), the dynamic programming is then to solve:

$$\min_{u_t\in U}\underset{W}{E}\left\{\sum_{t=0}^{\infty}\alpha^t g(i_t,w_t,u_t)\middle|i_0=i\right\},$$

and the value function is hence defined as

$$J(i_t)=\min_{u_t=U}\underset{w_t}{E}\left\{g(i_t,w_t,u_t)+\alpha J(i_{t+1}|i_t)\right\}.$$

This dynamic programming problem can be solved in theory using iteration algorithms discussed in Section 3.3.4. However, the computational requirement to compute $J(i)$ for all $i\in X$ in each stage can easily make the problem intractable. Approximation to dynamic programming reduces dimensionality and makes the problem computationally tractable. Appropriate approximation requires the reservation of fundamental properties of the value function. We discuss a couple of fundamental properties of the value function in the following section.

4.5 Properties of the Value Function

In this section, we prove some important structural properties of the value function in dynamic programming. The first property is the monotonicity of the value function in queue length. The second property is the monotonicity of the value function in controller state. The structural properties are essential for the development of approximation architectures.

4.5.1 Non-decreasing monotonicity in queue length

We now prove that the sufficient conditions (a), (b) and (c) of Theorem 3.3 are satisfied in the case of traffic signal control. We first look at the monotonicity of transition function $f(\cdot)$ defined by Eq. (4-17). We define that

$$i^+ \equiv \left[l^+, s, m_c, m_d\right], \text{ and } i^- \equiv \left[l^-, s, m_c, m_d\right], \tag{4-21}$$

so that

$$l^+ \succeq l^- \Leftrightarrow i^+ \succeq i^-. \tag{4-22}$$

Because that $f(\cdot)$ is a dependent on departure vector y, we prove the following property of y.

Lemma 4.1 *For all $u \in \{0,1\}$, and for all $i^+, i^- \in X$ such that $i^+ \succeq i^-$ we have*

$$y\left(l^+\right) - y\left(l^-\right) \preceq l^+ - l^-.$$

Proof: if $s_u(n) = 0$, from Eq. (4-13), (4-14) and (4-22), we have

$$\begin{aligned} y\left(l^+(n)\right) - y\left(l^-(n)\right) &= 0 \\ &\le l^+(n) - l^-(n). \end{aligned}$$

if $s_u(n) = 1$, at 5s resolution, by Eq. (4-13) we have

$$y\left(l^+(n)\right) - y\left(l^-(n)\right) \begin{cases} = l^+(n) - l^-(n) & \text{if } l^+(n) + w(n) < y^*(n) \\ < l^+(n) - l^-(n) & \text{if } l^+(n) + w(n) \ge y^*(n) \text{ and } l^-(n) + w(n) < y^*(n) \\ = 0 & \text{if } l^-(n) + w(n) > y^*(n). \end{cases}$$

Conditions at 2s and 0.5s resolutions can be proved similarly, except using Eq. (4-14) in place of (4-13), hence omitted. **q.e.d.**

The next lemma says that function $f(\cdot)$ is monopolistically non-decreasing in state i.

Lemma 4.2 *For all $u \in \{0,1\}$, and for all $i^+, i^- \in X$ such that $i^+ \succeq i^-$ we have*

$$f\left(i^+, w, u\right) \succeq f\left(i^-, w, u\right). \tag{4-23}$$

Proof: Given s, m_c, and m_d, for all $u \in \{0,1\}$, using Eq. (4-19), by Lemma 4.1 we have

$$\begin{aligned} f\left(i^+, w, u\right) - f\left(i^-, w, u\right) &= \left(l^+ - y\left(l^+\right)\right) - \left(l^- - y\left(l^-\right)\right) \\ &= \left(l^+ - l^-\right) - \left(y\left(l^+\right) - y\left(l^-\right)\right) \\ &\succeq \mathbf{0}. \end{aligned}$$

q.e.d.

Next, we show that the one-step cost function g is partially non-decreasing in state i.

Lemma 4.3 *The one-step cost* $g(i, w, u)$ *is partially non-decreasing in* $i \in X, \forall w$ *,* $\forall\ u \in \{0,1\}$*, and* $\forall\ t = 0, 1, \ldots, T-1$.

Proof: Let $i^+, i^- \in X$ such that $i^+ \succeq i^-$, from Lemma 4.2, and using Eq. (4-20), we have that

$$\begin{aligned} g\left(i^+, w, u\right) - g\left(i^-, w, u\right) &= \sum_{n=1}^{N}\left[\left(l^+ - y\left(l^+\right)\right) - \left(l^- - y\left(l^-\right)\right)\right]\Delta t \\ &= \sum_{n}\left[f\left(i^+, w, u\right) - f\left(i^-, w, u\right)\right]\Delta t \\ &\geq 0. \end{aligned}$$

q.e.d.

We further assume that the terminal cost function $h\ (i_m)$ is partially non-decreasing in i, thus all of the conditions for Theorem 3.3 (Section 3.5) are satisfied. From this, we have the following results:

Theorem 4.1 *The value function* J_t *for the traffic signal control problem is partially non-decreasing in vehicle queue length for all* $t = 0, 1, \ldots, T$.

Theorem 4.1 suggests that a linear function with non-negative functional parameters may work well in approximating the exact value function of traffic signal control.

4.5.2 Non-decreasing monotonicity in controller state

An intuitive reflection of traffic signal control is that vehicles queuing in the traffic link of red signal experience more delays than those in the link of green signal. In other words, value function J_t is partially non-decreasing if the signal vector s_u is partially switched from green to red. To facilitate further discussion, we define that

$$\begin{cases} s_u^+(n) \text{ red signal} \\ s_u^-(n) \text{ green signal.} \end{cases} \tag{4-24}$$

The following theorem states sufficient conditions for non-decreasing monotonicity of the J_t function in controller state.

Theorem 4.2 *Suppose the following conditions hold:*

(a) *For* $s_u^+(n) \equiv 1$, and $s_u^-(n) \equiv 0$ *we have* $f\left(i\left(s_u^+(n)\right), w, u\right) \succeq f\left(i\left(s_u^-(n)\right), w, u\right)$, $\forall u \in U$.

(b) *The one-step cost function* $g_t(\cdot)$ *satisfies* $g\left(i\left(s_u^+(n)\right), w, u\right) \geq g\left(i\left(s_u^-(n)\right), w, u\right)$, *for* $i \in X$,

and $\forall u \in U$, *t* = 0, 1, …, *m* − 1.

(c) *The terminal cost h*(i_m) *satisfies* $h\left(i_m\left(s_u^+(n)\right)\right) \geq h\left(i_m\left(s_u^-(n)\right)\right)$ *in* $i \in X$.

Then value function J_t *satisfies* $J_t\left(i\left(s_u^+(n)\right)\right) \geq J_t\left(i\left(s_u^-(n)\right)\right)$ *in* $i \in X$, *for t* = 0, 1, …, *m* − 1.

The proof of Theorem 4.2 is entirely similar to the proof of Theorem 3.3 (Section 3.5.1), hence omitted. We begin to show that this theorem holds in traffic signal control with the following lemmas.

The first lemma shows that less vehicle may possibly depart in red signal than in green.

Lemma 4.4 *For all* $i \in X$, $n = 1, 2, ..., N$, *for* $s_u^+(n)$ and $s_u^-(n)$ *defined by Eq.*4-24, *we have*

$$y\left(s_u^+(n)\right) \leq y\left(s_u^-(n)\right).$$

Proof: As defined by Eq. (4-13) and (4-14). **q.e.d.**

The next lemma says that red signal causes greater (or no less) transition cost than green signal.

Lemma 4.5 *For all* $i \in X$, $n = 1, 2, ..., N$, *for* $s_u^+(n)$ and $s_u^-(n)$ *defined by Eq.*4-24, $\forall u \in U$, *we have*

$$f\left(i\left(s_u^+(n)\right), w, u\right) \geq f\left(i\left(s_u^-(n)\right), w, u\right).$$

Proof: Given *s*, m_c, and m_d, $\forall u \in U, n = 1, 2, ..., N$, using Eq. (4-19), by Lemma 4.4 we have

$$\begin{aligned} f\left(i\left(s_u^+(n)\right), w, u\right) - f\left(i\left(s_u^-(n)\right), w, u\right) &= \left(l - y\left(s_u^+(n)\right)\right) - \left(l - y\left(s_u^-(n)\right)\right) \\ &= \left(y\left(s_u^-(n)\right) - y\left(s_u^+(n)\right)\right) \\ &\geq 0. \end{aligned}$$

q.e.d.

The next lemma says that red signal gives greater (or no less) one-step cost than green signal.

Lemma 4.6 *For all* $i \in X$, $n = 1,2,...,N$, *for* $s_u^+(n)$ and $s_u^-(n)$ *defined by Eq.*4-24, $\forall u \in U$, *we have*

$$g\left(i\left(s_u^+(n)\right), w, u\right) \geq g\left(i\left(s_u^-(n)\right), w, u\right).$$

Proof: From Lemma 4.5, and using Eq. (4-20), for any $n \in \{1,2,...,N\}$, we have

$$\begin{aligned} g\left(i\left(s_u^+(n)\right), w, u\right) - g\left(i\left(s_u^-(n)\right), w, u\right) &= \sum_{n=1}^{N} \left[\left(l(n) - y\left(s_u^+(n)\right)\right) - \left(l(n) - y\left(s_u^-(n)\right)\right)\right] \Delta t \\ &= \sum_{n=1}^{N} \left[f\left(i\left(s_u^+(n)\right), w, u\right) - f\left(i\left(s_u^+(n)\right), w, u\right)\right] \Delta t \\ &\geq 0. \end{aligned}$$

q.e.d.

Lemma 4.4 and 4.5 satisfy condition (a) of Theorem 4.2, Lemma 4.6 satisfies condition (b), and let condition (c) hold, we have met all of the conditions of Theorem 4.2. Therefore, we have:

Theorem 4.3 *The value function of traffic signal control problem formulated in dynamic programming is partially non-decreasing in controller state* s_u *such that*

$$J_t\left(i\left(s_u^+(n)\right)\right) \geq J_t\left(i\left(s_u^-(n)\right)\right).$$

4.6 Approximation to the Value Function

Theorem 4.3 suggests differentiation of controller status in the approximation function. Regarding this, we employ a feature-extraction function $\phi(i)$ such that,

$$\phi(i) = \begin{bmatrix} \phi\left(i(1)\right) \\ \vdots \\ \phi\left(i(N)\right) \end{bmatrix}, \tag{4-25}$$

where

$$\phi\left(i(n)\right) = \begin{cases} \begin{bmatrix} l(n) \\ 0 \end{bmatrix} \text{if } s(n) = 0 \\ \begin{bmatrix} 0 \\ l(n) \end{bmatrix} \text{if } s(n) = 1 \end{cases} \tag{4-26}$$

From Theorem 4.1, which postulates that value function in non-decreasing in traffic state, we propose a linear approximation function as described by the followings:

$$\tilde{J}(i,r) = \sum_{n=1}^{N} \phi'(i(n))r(n), \tag{4-27}$$

where

$$r(n) = \begin{bmatrix} r^{-}(n) \\ r^{+}(n) \end{bmatrix}. \tag{4-28}$$

By doing so, we assign r^- to queue length variable l (n) if link n receives green signal, or r^+ if otherwise.

The linear form of the approximation function allows the employment of many well-studied unsupervised learning techniques, such as temporal-difference learning (TD) and perturbation learning. In the next section, we discuss the general control policy for traffic signals.

4.7 Control Policies

A key fact underlying all dynamic programming methods is that *the only policies that are greedy with respect to their own evaluation function are optimal policies* (Barto *et al.*, 1995). Regarding this, we employ a greedy control policy to minimise vehicle delays at individual intersection. Because we only consider distributed control for traffic networks, the control policy for isolated intersection also applies for network operation. In this section, we limit our discussion on the general terms. The specific control policies are discussed specifically in Chapter 5 for each test case.

4.7.1 General control policy

The general control policy is to find decision u_t that minimises the sum of one-step costs for the planning horizon plus the future cost. Our immediate concern here is the length of planning horizon. Ideally, we would prefer to plan as much into the future as possible. However, real-time data from detectors are limited. In this study we assume that detectors provide 10s data of future vehicle arrivals (Assumption 4-6). For the horizon beyond the coverage of detected data, we seek to predict by using traffic models. Nevertheless, the further into the future, the less accurate the prediction would be. Robertson and Bretherton (1974) using backward DP showed that the optimal policy was not particularly sensitive to variations in the traffic arrivals of the next 10 to 20 seconds. Heydecker and Boardman (1999) further investigated a discrete time signal controller (at resolution 0.5s) in backward DP. The result showed that the optimal control sequences converge within a finite period of time, regardless of the initial states. These findings suggest keeping the planning horizon between 10s to 20s for real-time control. The actual length of horizon will be decided case-by-case in numerical studies.

4.7.2 Structure of the planning horizon

The part of the planning horizon with real-time data from detectors is regarded as the 'head' of the horizon, whilst the part with predicted data being the 'tail'. There are a variety of approaches to supply predicted data for the tail of the horizon. Gartner (1983a) proposed four types of tail models, which are *variable tail, fixed tail, static tail* and *dynamic tail*.

In the variable tail model, we project upstream arrivals for the head period as well as for the tail period, i.e., we assume to have accurate information on arrivals for the entire horizon. This model will be modified and used for the test on network operation in this study.

In the fixed tail model, we predict the future arrivals by running a smoothing process on the arrival data. Consequently, the tail values will slowly vary with the moving average of the flow rate. While it causes some degradation in performance when compared with the variable tail, this model is suitable for on-line implementation since it only requires data that are readily available from detectors. In this study, the fixed tail model will be used for the test on isolated intersection.

Static tail and dynamic tail models deal with cyclic patterns of arrivals for the entire data set. The two models are more appropriate for theoretical investigations, as they require the projection of arrivals for the entire length of the control period, and have strong cyclic pattern assumption, which is less relevant to the interest of our study.

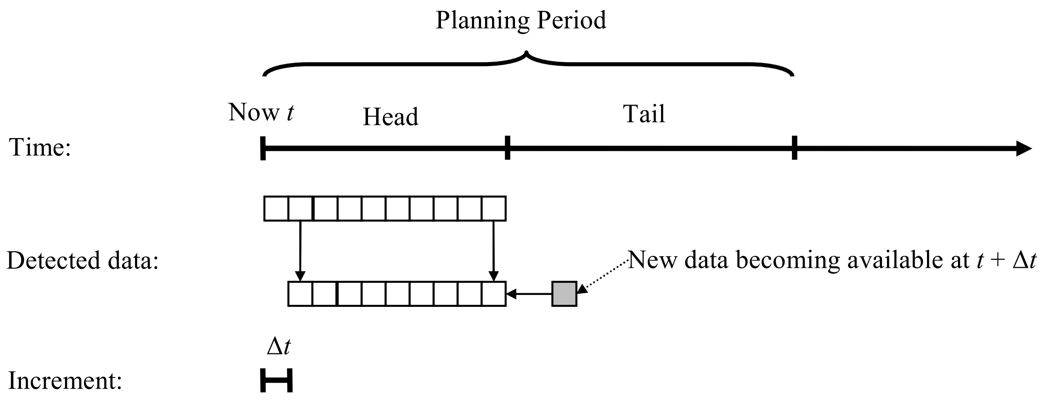


Fig. 4-1 The rolling horizon approach and the emergence of detected data of future traffic arrivals

4.7.3 Rolling horizon

Under Assumption 3-1 and 4-6, we have *new* data of Δt at the beginning of each time interval. This process is illustrated in Fig. 4-1. To utilise the newly available data, we first calculate a detailed signal plan for the planning horizon, which includes both the head and tail parts, and then only the first Δt of each such plan is implemented: the rest is revised in the arrival of new detected data. This is a rolling horizon approach, which we have described concisely in Section 2.2.3. Using the rolling horizon, the smaller the time increment Δt is, the more frequent the signal plan is revised. For example, at the 0.5s resolution, the signal plan is revised 10 times more often than at the 5.0s resolution. We expect that a fine resolution would bring additional reduction in vehicle delays because of the greater flexibility in revising plans to accommodate dynamics in traffic.

4.7.4 The *M*-step iteration

In general, we can assume that there are M temporal steps in total during the horizon. The value of M may vary according to the actual length of the horizon and the resolution of the discrete system. Recalling the M-transition Bellman's equation introduced in Section 3.4.5, we have

$$J(i)=E\left[\sum_{t=0}^{M-1}\alpha^t g(i_t,i_{t+1})+\alpha^M J(i_M)\middle|i_0=i\right],\ i\in X\ .$$

Using Eq. (4-20) to substitute the more general term $g\ (i_t,\ i_{t+1})$, we have

$$J(i)=E\left[\sum_{t=0}^{M-1}\alpha^t g(i_t,w_t,u_t)+\alpha^M J(i_M)\middle|i_0=i\right],\ i\in X\ . \qquad (4\text{-}29)$$

The value iteration method corresponding to this modified form is

$$J_t(i)=E\left[\sum_{k=t}^{t+M-1}\alpha^t g(i_k,w_k,u_k)+\alpha^M J_{t-1}(i_{t+M-1})\middle|i_0=i\right],\ i\in X\ . \qquad (4\text{-}30)$$

Substituting exact J_t with the approximate function $\tilde{J}_t$ defined by Eq. (4-27), the greedy control policy is to find

$$\mathbf{u}_t^*=\arg\min_{u_t\in U}E\left[\sum_{k=t}^{t+M-1}\alpha^t g(i_k,w_k,u_k)+\alpha^M\tilde{J}_{t-1}(i_{t+M-1},r_{t-1})\right],\ i\in X\ , \qquad (4\text{-}31)$$

where $\mathbf{u}_t^*$ is a $N\times M$ vector whose k^{th} column is equal to u_k; that is

$$\mathbf{u}_t^*=\begin{bmatrix} | & & | \\ u_t & \cdots & u_{t+M-1} \\ | & & | \end{bmatrix}=\begin{bmatrix} - & u_k(1) & - \\ & \vdots & \\ - & u_k(N) & - \end{bmatrix}, \qquad (4\text{-}32)$$

where N is the total number of links of the traffic intersection.

The decision space of the control policy includes:

(a) The signals are not changed;

(b) The signals change immediately to the signal stage that gives least total delay.

The control process using the M-step iteration method and under Assumption 4-6 and 4-7 can be expressed as:

$$\mathfrak{F}=\left\{i_0,w_0,w_1,\ldots,w_{M-1},u_0^*;i_1,w_1,w_2,\ldots,w_M,u_1^*;i_2,\ldots;i_{m-1},w_{m-1},w_m,\ldots,w_{m+M-1},u_{m-1}^*;i_m\right\}.$$

Implementing option (b) may be further subject to specific terms, and the terms may vary from isolated intersection to networks. We will discuss the specific terms together with the numerical tests in Chapter 5.

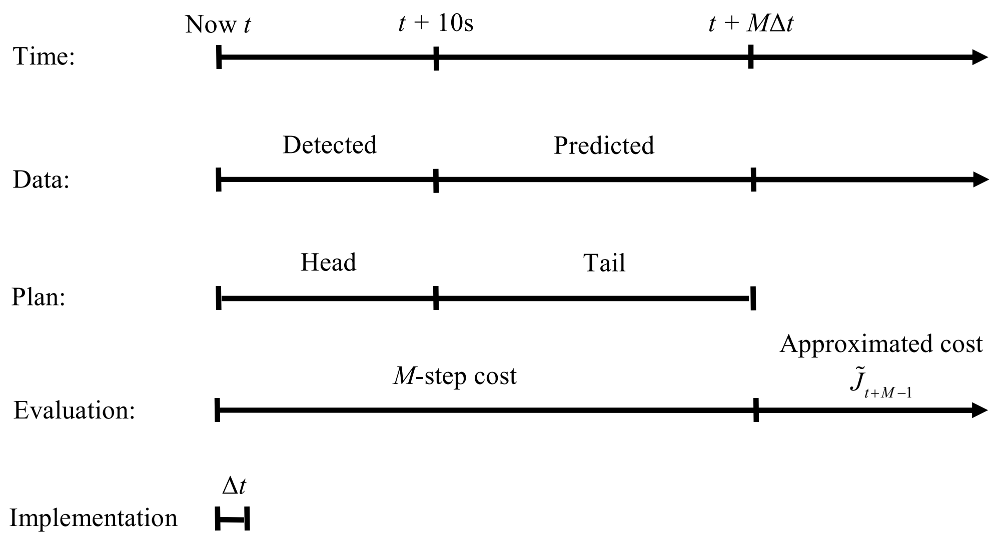


Fig. 4-2 The general control policy and a rolling horizon approach; A planning horizon consists "head" and "tail" parts, only the first Δt part of the signal plan is implemented.

According to the rolling horizon concept, only the first Δt part of the signal plan is actually implemented. The system then rolls forwards into the next temporal interval. This process is illustrated in Fig. 4-2.

4.7.5 Summary

The general control policy aims to minimise the sum of immediate vehicular delays in a specified planning horizon and the future delays. The horizon can be divided into head part and tail part. The head part is supplied with real-time data from detectors, and the tail part supplied with predicted data from traffic models. Signal plans, however, are only implemented for the first Δt of the planning period. The system then rolls forward and revises plan as new data emerge. The finer the resolution of Δt, the more frequent the signal plan is revised.

4.8 Traffic Models

Traffic models serve for two purposes in our study:

1) It simulates traffic dynamics at signalised traffic intersections and in the links of traffic networks
2) It assists to evaluate control decisions.

We begin our discussion with the fundamentals of modelling traffic.

4.8.1 Fundamentals

Traffic models in general are concerned with finding relationships between the three fundamental variables of traffic: flow q, speed V[1] and density K. The three characteristic variables describe the average behaviour of traffic flow over different locations and different observation periods. The fundamental relationship between these three variables is established by Lighthill and Whitham (1955), and independently by Richards (1956), and is described as

$$q = VK \,, \tag{4-33}$$

which frequently referred as the LWR model. From this model, a number of fundamental traffic flow theories, such as the propagation of shockwaves, are derived.

The LWR model is a macroscopic approach, in which the behaviour of individual vehicles cannot be distinguished. Lighthill and Whitham further assumed that q and K are related in a fashion described by what come to be known as the "Fundamental Diagram", as shown in Fig. 4-3. Supported by the empirical evidence, the LWR model is arguablely one of the most widely accepted models of traffic flow at macro level. It forms the premise for both advanced analytical studies and practical approximations.

Modelling traffic is an established field in transport studies. A good guide to traffic flow theories and traffic modelling is available in Gartner *et al*. (1997). In this section, we only discuss models conforming to our discrete, acyclic, and distributive control system. Discussions are divided into isolated intersection and traffic networks.

[1] We use upper case V here to denote flow speed rather than the more conventional lower case v because that the latter is used in Section 3.4.1 for denoting the *induced local field* of a neuron model.

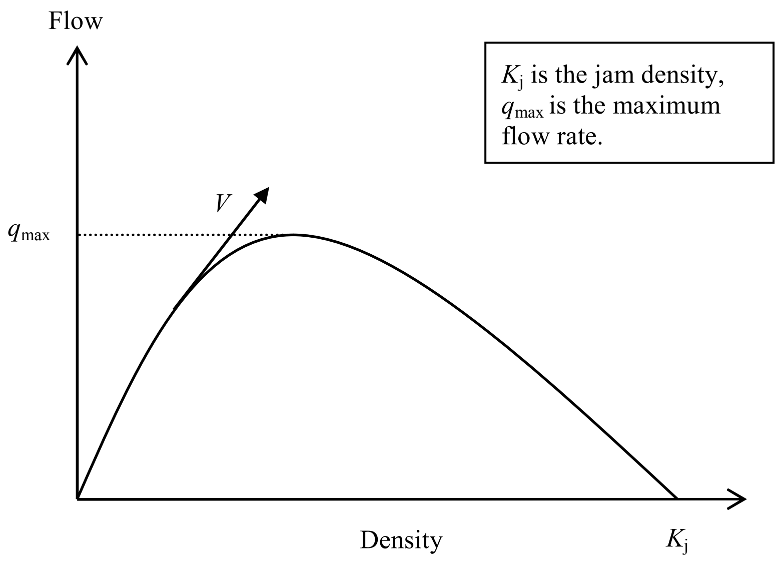


Fig. 4-3 The Fundamental Diagram of the LWR model

4.8.2 Isolated intersection

In this case traffic model is limited only to describe the traffic dynamics of the intersection itself, regardless of the upstream and down stream effects. The key variables are queue length *l*, the arriving vehicles *w*, and the departing vehicles *y*. Their relationship is regulated by Eq. (4-16). We further assume that the arriving vehicles are travelling at homogenous speed from the upstream detector line to stop-line, where vehicles join a queue vertically. The physical length of the queue is neglected in the model; therefore the queue does not spill back to the upstream. Once vehicles are held in a vertical queue, the departure time of the first vehicle is assumed to coincide with the start of effective green. Since in our study we neglect lost time (Assumption 4-5), the start of effective green is equivalent to the start of green signal. The following vehicles are assumed to depart from the stop-line at equal headways (saturation departure time) until queues are dissipated, as it is regulated by Eq. (4-13) and (4-14). This dynamic system automatically forms a vertical queuing model, variations of which are widely adopted in traffic signal optimization tools, such as DYPIC OPAC and TRANSYT. The vertical queuing model is simple in comparison with other traffic models, as it assumes that all queuing vehicles move with the same speed, and stop instantaneously.

Let q denote the traffic flow rate in a selected link, y^* the saturation departure rate, T_r the period of red signal, and T_g the period of green signal, we can plot the queue formulation and dissipation in vertical queuing model against time, as shown in Fig. 4-4. The queue length accumulated in T_r is obtained by

$$l = qT_r,$$

and the green time to dissipate the queue length by

$$T_g = \frac{qT_r}{\left(y^* - q\right)}.$$

Examples of the actual queue formulation and dissipation in simulation are presented in Fig. 4-5 (a) under 5s resolution and in Fig. 4-5 (b) under 0.5s resolution. The vertex of the triangles in Fig. 4-5s is right-shifted in comparison with the position in Fig. 4-4, which shows that the rate of dissipation is greater than the rate of arrival. It is worth noting that in Fig. 4-5 (b) the dissipation rate in green period can temporarily become positive. This is because that under the resolution of 0.5s, during the headway of the last departure, an additional vehicle joins the queue. This does not apply to the 5s resolutions, as regulated by Eq. (4-13).

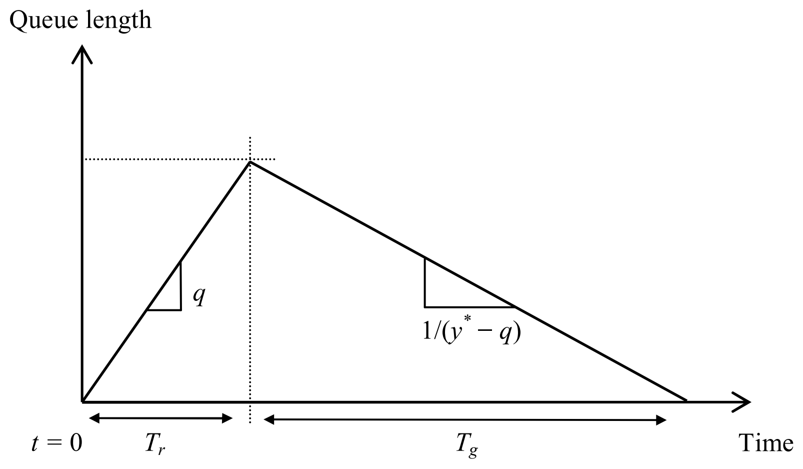


Fig. 4-4 Queue formation and dissipation in the vertical queuing model; T_r is the red signal period, T_g the green signal period, q the traffic flow rate, and s the saturation departure rate

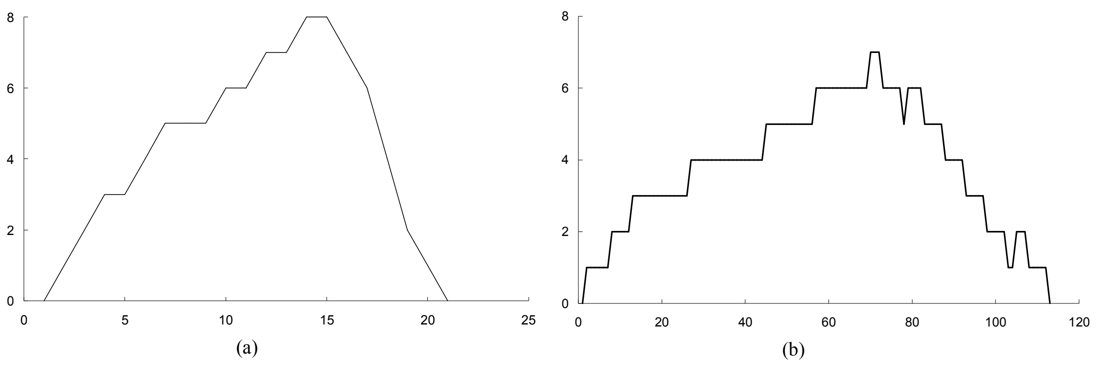


Fig. 4-5 Actual formulation and dissipation of queue in the vertical queuing model in simulations (x-axis: time interval; y-axis: queue length); (a) shows queue formation at 5.0s resolution, and (b) at 0.5s.

The vertical queuing model, however, is not the only traffic model that can be possibly applied in discrete control system. Ahn (2004) investigated a microscopic car-following model in the context of a discrete dynamic signal controller. His work on dynamic controller shares similarity with ours, despite the fact that his work assumes a fixed stage sequence, and the controller is constrained by the maximum green time. Ahn concluded that, although the microscopic is more precise in modelling vehicle motion and in calculating vehicle delays, the vertical queuing model is nearly as good as the microscopic car-following model when used for signal timing optimisation. This conclusion suggests keeping the model simple. We therefore adopt the vertical queuing model in our study on isolated intersections.

4.8.3 Traffic network

In a traffic network, traffic dynamics in adjacent intersections are inter-correlated. The outflow of traffic from the upstream intersection influences the formulation and dissipation of queue in the downstream intersection. Similarly, queues of downstream may influence the outflow traffic of the upstream. For example, in a short traffic link, queues of downstream may spill back to the stopline of the upstream, blocking the vehicles that would have been departing. Therefore, it requires a traffic model for the entire network to describe the inter-correlated flow dynamics. We review *platoon-dispersion model* and *cell transmission model* in the following discussion.

4.8.3.1 Platoon dispersion model

A direct extension of the vertical queuing model to network study is the *platoon dispersion* model (Robertson, 1969) used in TRANSYT. This model is developed under the assumptions that the signal timings are fixed and cyclic, and thus the flows along the links are cyclic. This model calculates the behaviour of traffic by manipulating the following three types of cyclic flow profiles:

1. IN profile: the arrival profile at the down stream stopline, if the traffic were not impeded by the signal at the stopline.
2. GO profile: the profile of traffic that would leave the stopline, if there was enough traffic to saturate the green.
3. OUT profile: the departure profile of traffic actually leaving the stop-line; it is actually equal to the GO profile as long as there is a queue; after the queue has discharged, it is equal to the IN profile for the duration of the effective green time.

To model platoon dispersion (and therefore for all calculations of the flow patterns), the common cycle plan of the network is divided into *K* number of intervals. Profiles are then calculated as step functions with a stepsize of one interval. The number of intervals is often taken as the number of seconds in the cycle, so that an interval has duration of 1 second.

Let q_k and q'_k be the amount of traffic in interval *k* in the upstream and downstream In-profiles respectively, then $q_{k+K} = q_k$ for all *k*, and similarly for q'_k in terms of various q_k. The platoon dispersion model assumes that each q_k contributes $\mathrm{F}(1-\mathrm{F})^n q_k$ to each $q'_{(k+t+n)(\mathrm{mod}\ K)}$ for $n = 0, 1, 2, \ldots$, where *t* is an integer determined by the travel time along the link and F is a proportion. It follows that

$$q'_{(k+t+n)(\mathrm{mod}\ K)} = \mathrm{F}q_k + \left(1-\mathrm{F}\right)q'_{(k+t-1)(\mathrm{mod}\ K)},\ k = 1,2,\ldots,K\,. \tag{4-34}$$

Traffic is conserved because $\sum_{t=0}^{\infty}\left(1-\mathrm{F}\right)^t = 1/\mathrm{F}$.

Although the platoon dispersion model is one of the frequently used tools for signal optimisation, its assumption of cyclic flow pattern and the specified relationship of Eq. (4-34) do not hold in acyclic control. This is because the local controller may not return to the same signal stage after *K* intervals, nor follow the same stage sequence. Consequently, there is no guarantee of a cyclic flow pattern after all. Furthermore, the platoon dispersion model is designed for a centralised network control strategy, in which a common cyclic time applies to all local intersections, except for the settings of offset and specific start/end of green phases.

Since our interest lies in distributed control strategy with acyclic signal timings, we look into traffic models that describe link flow independent of the assumption of cyclic profiles and common cycle plans.

4.8.3.2 Cell transmission model

In discrete systems, we only need to consider the distribution of vehicles in a traffic link at successive temporal intervals. To make the model at macro level, the distribution of vehicles in the link may be only at state of ‘free flows’ or a ‘queue’ during a temporal interval. The traffic link can be segmented into an array of ‘blocks’, as shown in Fig. 4-6, and the distribution of vehicle is either travelling at free flow speed from one block to the next, or, joining a queue and remains in the current block. Therefore, once a vehicle depart from upstream intersection and enters into the rear block of the link leading to downstream, its distribution will be processed among the blocks successively, until departing from the downstream stopline.

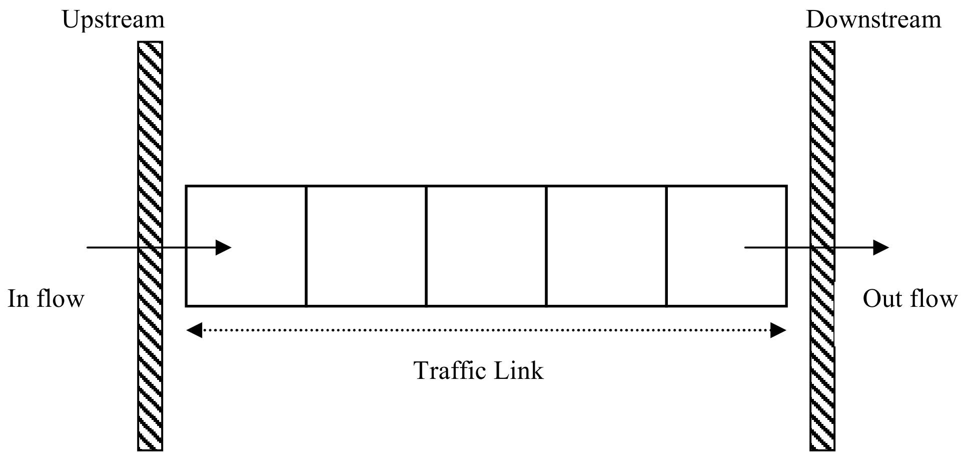


Fig. 4-6 A graphical presentation of segmented traffic link

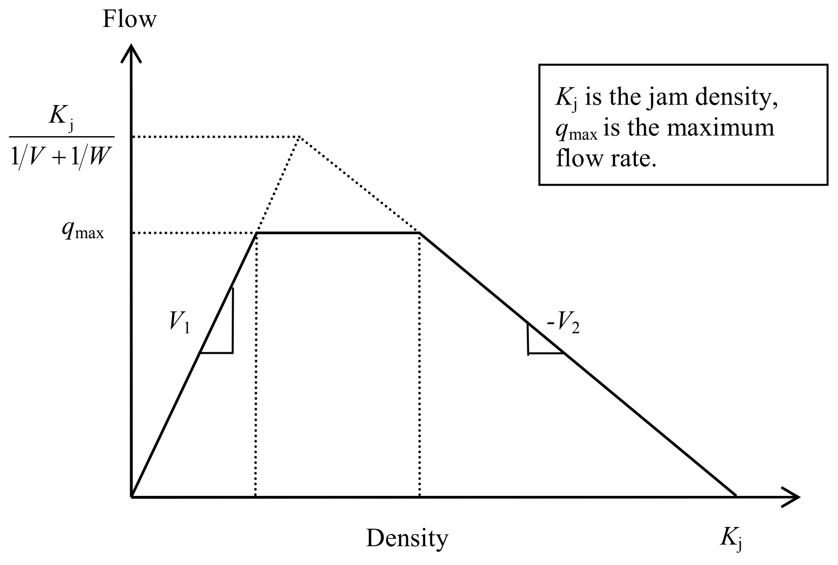


Fig. 4-7 Flow-density relationship for the generalised cell-transmission model

These basic ideas were explored in specific ways by Gartner (1983a) in developing OPAC and by Henry *et al.* (1983) in developing PRODYN. These ideas are generalised in Daganzo (1994), who introduced a macro traffic flow model called *Cell Transmission Model* (CTM).

The CTM model is a discrete approximation of the LWR model. It assumes its fundamental diagram to take a triangular or trapezoidal form as shown in Fig. 4-7. This relationship assumes a constant free-flow speed, V_1, for lower densities and a constant negative wave speed, $-V_2$ (always lower than free-flow speed) at higher densities.

In the cell transmission model, a traffic link is represented by a collection of equal-length cells. The length of each cell is equal to the distance that a single vehicle travels during one temporal interval at the free-flow speed. If there is no congestion, it is expected that a vehicle would move from one cell to another during the interval. For a given time interval t, each cell k has $x_t\,(k)$ number of vehicles and $y_t\,(k)$ vehicles ready to enter. The outflow from each cell k (or the inflow into its downstream cell $k-1$) during the time interval t to $t+\Delta t$ is governed by

$$y_t(k-1) = \min\left\{x_t(k), Q(k-1), \frac{V_2}{V_1}\left[C(k-1) - x_t(k-1)\right]\right\} \qquad (4\text{-}35)$$

where

Q (*k*) is the maximum number of vehicles that can enter cell *k* in a single time interval;

C (*k*) is the spatial capacity of cell *k*;

$\left[C(k-1) - x_t(k-1)\right]$ is the available space in cell $k-1$; and

$\frac{V_2}{V_1}$ is the ratio of shockwave speed to free-flow speed.

Eq. (4-35) models both the congested and uncongested regions through the fundamental diagram. After the outflows are determined for each cell for a specified time interval, the traffic conditions in the network at the next time interval, $t+1$, are updated using the following conservation equation:

$$x_{t+1}(k) = x_t(k) - y_t(k-1) + y_t(k). \qquad (4\text{-}36)$$

The CTM model is embedded in Lo's (1999, 2001) integer programming solution to network traffic signal control, in which the exit flow capacity of a signal cell is defined and controlled by the signal settings. The signal controller exhaustively searches the duration of green phases between the boundaries of maximum and minimum time constraints to optimise performance index. Its implication limits to cyclic signal plans. Lo *et al.* (2001) used CTM model to support a dynamic signal optimisation heuristic based on genetic algorithm. The heuristic was tested in real sites in Hong Kong and was proven competitive in performance in comparison with TRANSYT plans. The cyclic signal plans are generated randomly for all the intersections of the network, and then are evaluated and regenerated by the genetic algorithm. The disadvantage of this approach is that the understanding of the heuristic is largely kept at black-box level because of the genetic algorithm. Wong *et al.* (2007) employed the CTM model to study the reserve capacity of signalised traffic network. They formulated the control problem using a binary-mix-integer-program, which was then solved by a standard branch-

and-bound technique. The signal timings are acyclic, although the signals of the example junction are optimised together rather than distributedly.

Given the maturity and conformity of the CTM model to discrete and dynamic signal control problem, we use this model to describe the traffic dynamics in our test on network control.

4.8.4 Summary

In this section we examined a few traffic models for their compatibility and conformity to our study. We found that the vertical queuing model is simple and sufficient for modelling traffic at isolated intersection and the CTM model for traffic network. Both models are mature and widely adopted in signal optimisation studies. The purposes of traffic models are to simulate traffic dynamics in numerical experiment and assist evaluation of control decisions.

4.9 The ADP Algorithms for Traffic Signal Control

In the preceding sections of this chapter, we have introduced in sequential order the random generation of traffic demand for numerical experiment, the dynamics of state transition, the key properties of the value function, the general control policy and the traffic models. Together with the ADP formulae and learning techniques introduced in Chapter 3, we are now able to develop ADP algorithms for adaptive traffic signal control. Discussions in this section is divided into isolated intersection and traffic networks.

4.9.1 ADP algorithm for isolated intersection

In Chapter 3 we discussed two alternative learning techniques for updating the linear approximation function. One is the temporal-difference (TD) method, and the other is the perturbation learning (PL) method that numerically calculates partial gradients. In the rest of the thesis, we denote the former method as the ADP_TD option, and the latter as the ADP_PL option. Since the primary difference of the two optional techniques lies in the mechanism in generating learning signals and the routine in updating functional parameters, we propose a single ADP algorithm, while highlighting the two optional techniques in the process of updating functional parameters.

We further define that the isolated intersection has multiple signal stages and has N traffic links in total. The control algorithm is applicable to all resolutions in concern. The ADP algorithm is described as follows.

Step 0: Initialisation

0.1 Choose an initial system state i_0 ;

0.2 a) ADP_TD option, initialise neural network settings and neural weight vector r_0;

b) ADP_PL option, choose initials values for functional parameter vector r_0;

0.3 Initialise learning rate (or stepsize) 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 information w_t for the head part of the planning horizon;

1.3 Predict the information w_t' for the tail part of the planning horizon, if applicable.

Step 2: Evaluate control decisions

2.1 If switch constraint $m_c > 0$, signal change is not admissible, and $\mathbf{u}^*_t = \mathbf{0}$,

set $m_c = \max\{0, m_c - 1\}$;

2.2 If switch constraint $m_c = 0$, for the planning horizon of M-steps, find the optimal decision $\mathbf{u}^*_t$ using Eq. (4-31), i.e.

$$\mathbf{u}_t^* = \arg\min E\left[\sum_{k=t}^{t+M-1} \alpha^t g\left(i_k, w_k, u_k\right) + \alpha^M \tilde{J}_{t-1}\left(i_{t+M-1}\right)\right],\ i \in X .$$

if the optimal decision is to switch signal, then set $m_c = M_c$, where

$$\left(M_c + 1\right)\Delta t = T_{\min} + T_{\text{inter}} ;$$

if the optimal decision is to remain current signal indication, set $m_c = \max\{0, m_c - 1\}$.

Step 3: Update approximation function

a) ADP_TD option:

3.a.1 Using $\mathbf{u}^*_t$ and Eq. (4-30), calculate new observation $\hat{J}_t(i_t)$, i.e.

$$\hat{J}_t(i_t) = E\left[\sum_{k=t}^{t+M-1} \alpha^k g(i_k, w_k, u_k^*) + \alpha^M \tilde{J}_{t-1}(i_{t+M-1})\right],\ i \in X\ ;$$

3.a.2 Calculate current approximation $\tilde{J}_{t-1}(i_t, r_{t-1})$ by using Eq. (4-27), i.e.

$$\tilde{J}_{t-1}(i_t, r_{t-1}) = \sum_{n=1}^{N} \phi'(i_t(n)) r_{t-1}(n),\ n = 1, 2, \ldots, N;$$

3.a.3 Calculate M-step temporal difference

$$\sum_{k=t}^{t+M-1} \alpha^{k-t} d_k(i_k, i_{k+1}) = \hat{J}_t(i_t) - \tilde{J}_{t-1}(i_t, r_{t-1});$$

3.a.4 Update functional parameter vector r_{t-1} using Eq. (3-74), i.e.

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

b) ADP_PL 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 Δl,

if $s(n) = 0$ (green signal in link n), using Eq. (3-81),

$$\Delta r_t^-(n) = \frac{\hat{J}_t\big(i_t(l_t(n) + \Delta l(n))\big) - \hat{J}_t(i_t)}{\|\Delta l(n)\|},$$
$$\Delta r_t^+(n) = 0;$$

if $s(n) = 1$ (red signal in link n), using Eq. (3-81),

$$\Delta r_t^+(n) = \frac{\hat{J}_t\big(i_t(l_t(n) + \Delta l(n))\big) - \hat{J}_t(i_t)}{\|\Delta l(n)\|},$$
$$\Delta r_t^-(n) = 0;$$

where

$$\hat{J}_t(i_t) = E\left[\sum_{k=t}^{t+M-1} \alpha^k g(i_k, w_k, u_k^*) + \alpha^M \tilde{J}_{t-1}(i_{t+M-1})\right],\ i \in X\ ,$$

and

$$\tilde{J}_{t-1}(i_t, r_{t-1}) = \sum_{n=1}^{N} \phi'(i_t(n)) r_{t-1}(n),$$

3.b.2 Update functional parameter vector r_{t-1} using Eq. (3-82), and for $n = 1, 2, \ldots, N$,

if $s(n) = 0$ (green signal in link n)

$$\begin{aligned} r_t^-(n) &= \left(1-\eta_k^-(n)\right) r_{t-1}^-(n) + \eta_k^-(n)\Delta r_t^-(n) \\ r_t^+(n) &= r_{t-1}^+(n), \end{aligned}$$

if $s(n) = 1$ (red signal in link n)

$$\begin{aligned} r_t^+(n) &= \left(1-\eta_k^+(n)\right) r_{t-1}^+(n) + \eta_k^+(n)\Delta r_t^+(n) \\ r_t^-(n) &= r_{t-1}^-(n), \end{aligned}$$

where $\eta_k^-(n)$ and $\eta_k^+(n)$ are the stepsizes for $r_t^-(n)$ and $r_t^+(n)$ respectively, and k denotes the number of times the parameter has been updated.

Step 4: Implement optimal decision $\mathbf{u}^*_t$ for the first Δt of the planning period

4.1 Transfer signal status using Eq. (4-4) and (4-15);

4.2 Transfer queue status using Eq. (4-16);

4.3 Record switch constraint m_c and departure constraint m_d.

4.4 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.

Using ADP_TD, from Theorem 3.2 (Section 3.4.4, Chapter 3) we know that if the assumptions hold, the parameter vector of the linear approximation function converges with probability of 1. Theorem 4.1, 4.2, and 4.3 guarantee that ADP_PL preserves the structural properties of the value function.

4.9.2 ADP algorithm for traffic network

In traffic network, a control decision at local intersection influences the traffic in connected traffic links and at adjacent intersections. Using CTM model we are able to evaluate the influence of local decision to adjacent intersections at macro level. The concept of distributed control requires signals be optimised locally, without centralised or collective optimisation. This means that microprocessors embedded in local controllers calculate in parallel rather than following preset sequential order. Moreover, communications may be established among adjacent controllers to exchange the information of the current local system state, including queuing state and controller state.

Existing network control systems usually limit the functionality of a local controller. OPAC decomposes a network into sub-networks, and at each time instant selects a critical intersection in the sub-network to optimise signal timing, while preserving current timing plans in the rest intersections of the sub-network. Sub-networks may overlap, but optimisations in selected intersections are parallel. PRODYN adds one supervisory layer on top of the local intersections in the network topology. Local decisions are submitted to the supervisory layer for review and receive dispatched sensitivity variables in return for further optimisation. This process carries on until no better combination of local decisions is found.

In this study, optimisations at intersections are completely parallel and independent. Each local ADP controller requires an individual approximation function. When the local controller is evaluating control decisions, the controller state of other intersections is assumed unchanged. The traffic states are interpreted from the CTM model. The control algorithm for each local controller is the same as described in Section 4.9.1.

4.9.3 Summary

In this section, we introduced the ADP control algorithms for isolated intersection and traffic network respectively. The algorithms adopt a general framework that accommodates both temporal-difference learning and perturbation learning. The control of traffic networks is fully distributed, and control algorithm at each local intersection is the same as for isolated intersection control.

4.10 Discussion

This chapter describes the development of the ADP algorithm for adaptive traffic signal control. After introducing a few general assumptions for the traffic signal control problem, we introduced the formulae that describe the fundamental relationships between queue, signal, arriving vehicle and control decisions. The state transition formulae are the basis for proving theorem 4.1, 4.2, and 4.3 that identify structural properties of the value function. Based on these properties, a linear approximation function using feature-extraction function is proposed.

We proposed in this chapter a general control policy that is greedy in respect to the evaluation function. There is no assumption of pre-set cyclic plans or stage sequence. The planning horizon is divided into the head part and the tail part. The former is supplied with detected information, and the latter with predicted information. The optional decisions are evaluated by using traffic models that simulate the system ahead. We identified that vertical queuing model is sufficient for modelling traffic dynamics at an isolated intersection, and the Cell Transmission Model (CTM) model for modelling traffic network. Using the rolling horizon approach, only the first Δt part of the signal plans is implemented. The system then rolls forward one step.

The ADP algorithm here developed is applicable for both isolated intersection and traffic networks. The differences are that, in traffic network, local decisions influence the downstream traffic, and CTM model is used to model dynamics in traffic links as well as at the stoplines. The coordination between intersections is implicit in the distributed control method. Influence of local decision to downstream is processed by traffic models, and state of local control system is interpreted from the traffic models.

The design of the algorithm reflects the practical control condition. The assumption of 10s information of future vehicle arrivals is realistic in contemporary traffic engineering. Modern signalised intersection, such as those controlled by MOVA, uses inductive loops installed upstream to provide information of several seconds' future arrivals. A vehicle passing over the loop generates impulse to the detector, which then interpret the impulse as "1" or "0". This is fairly similar to the definition of demand vector w_t in this study. By using the rolling horizon approach, the signal timing plans calculated at the preceding time interval are revised at the frequency determined by the resolution. OPAC and PRODYN work at the resolution of 5s, and MOVA at 0.5s. The design of the ADP algorithm accommodates a range of resolution from 5s to 0.5s. The computational advantages of ADP algorithms raise the prospect of working at even finer resolutions. Above all, the ADP algorithm operates with minimum human intervention, as the controller uses learning mechanisms to adapt to changing traffic environment. The ADP algorithms do not require significant upgrade of existing computing hardware and communication facilities.

However, necessary simplifications are made to facilitate investigations. The modelling of traffic dynamics is primarily at macro level. Consequently, the individual vehicle behaviours during a temporal interval are neglected. Even though we may adopt a rather fine resolution for the discrete time system, the description of vehicle behaviour is limited to either travelling at free-flow speed or remaining in a queue. In contrast, a microscopic traffic model, such as car-following model, is capable of modelling de/accelerations of individual vehicles, or even vehicle taking-over and lane changing. Implications of using microscopic traffic model into ADP controller are left for further studies.

Furthermore, traffics are assumed as homogenous. The models in use do not differentiate vehicles by their physical characters, speed or utility. More sophisticated models, together with further development in control algorithm, are required to address issues like bus priority or other specific control schemes.

In addition, this study neglects amber stage of a traffic signal and the lost time of green phase. Pedestrian phases are not considered. Those simplifications allow us to focus on addressing the primary concerns of traffic signal control in the development of ADP algorithm, i.e. minimising road user delays. We may extend the ADP algorithm to more complex environment on the condition that it performances well in relatively simple cases.

In the next chapter, we present numerical experiments in using ADP method for traffic signal control.

CHAPTER 5 NUMERICAL EXPERIMENTS

This chapter presents a systematic investigation into the performance of the ADP controllers through simulation-based experiments. The scope of these experiments encompasses both isolated intersection and small-scale traffic networks. The objectives of the experiments are explained in Section 5.1. Experiments on controlling isolated intersections are presented in Section 5.2, and those on traffic network operations in Section 5.3. Conclusions drawn from experiment results are presented in Section 5.4.

5.1 Objectives

Objectives of conducting numerical experiments are to:

1) Test performance of ADP controllers against benchmarks;
2) Test performance of ADP controller at different temporal resolutions of the discrete time process;
3) Investigate evolution of the approximation;
4) Evaluate effects of reinforcement learning on control performance.

The starting case for the numerical experiments is a two-arm, two-stage road intersection. This simple layout has been frequently employed as test-bed for various control methods, thus offering good background for performance comparison.

In the second stage of experiment, we extend the geometric layout to intersections with multiple signal stage control. Considering multi-stage intersection facilitates, we compare between acyclic and cyclic controllers. Furthermore, in order to investigate effects of different temporal resolutions on controller performance, we compare performance between coarse and fine temporal resolutions.

In the final stage of experiment, we implement ADP controllers for operation in a small-scale traffic network. Signal operation in the network is distributed, with each intersection governed by a local ADP controller. Without preset rules for signal coordination, we investigate whether this distributed control strategy can yield competitive results. The cell transmission model (CTM, Daganzo, 1994) is used for modelling traffic dynamics in the network.

5.2 Isolated Intersection

Experiments at isolated intersection encompass two cases: a two-arm and two-stage case, and a three-arm and three-stage case. In both cases, each arm has a single link, and each link has a single traffic lane. The general assumptions introduced in Section 4.1 apply in both cases. Random traffic is generated according to the algorithms and rules introduced in Section 4.2. System dynamics and ADP algorithms are as described in corresponding sections of Chapter 4, except for wherever specified in the experiment.

5.2.1 Two-stage intersection

Numerical results in controlling a two-stage intersection with ADP_PL system were first presented in Cai (2007). The intersection layout is shown in Fig. 5-1 (a). It has a major traffic flow from east to west on Link A, and a minor flow from north to south on Link B. Stage sequence for fixed-time planning is shown in Fig. 5-1 (b). In Stage 1, Link A receives green signal, while Link B receives red signal; signal indications are reversed in Stage 2. Resolution of the discrete time system is set at 5s per time increment. By using rolling horizon approach, the controller revises the signal plan every 5s.

The controller is tested under a range of traffic inflows that are summarised in Table 5-1. The combination of inflows represents typical urban traffic demand. Each run of experiment simulates 600 seconds, equivalent to 120 time intervals at 5s resolution.

Table 5-1 Traffic flows (vehicles per hour) combinations for the two-stage intersection

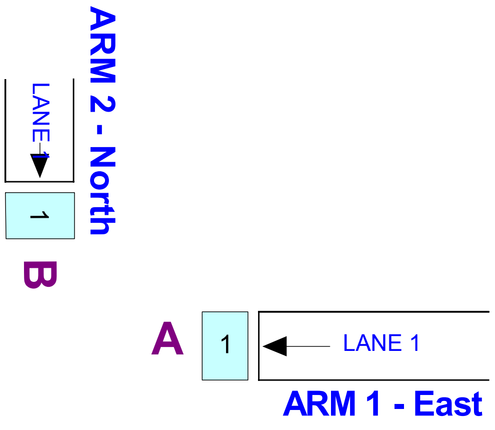
Link B	Link A			
240	252	396	600	678
432				

Only the ADP_PL mode is employed in this experiment. This early study on ADP_PL controller differs from the later cases in that we use a simpler approximation structure. In this case we define the feature-extraction function as

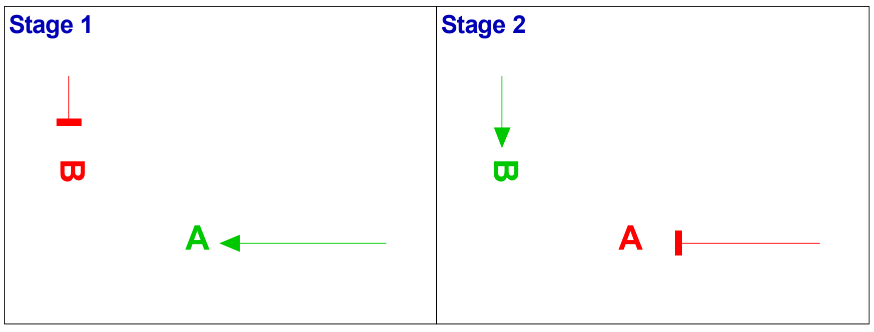
$$\phi(i) = \begin{cases} \begin{bmatrix} l_A \\ l_B \end{bmatrix} & \text{if } s_A = 1 \\ \begin{bmatrix} l_B \\ l_A \end{bmatrix} & \text{if } s_B = 1, \end{cases} \tag{5-1}$$

and function parameter vector as

$$r = \begin{bmatrix} r^- \\ r^+ \end{bmatrix}, \tag{5-2}$$



5-1 (a) Geometric layout of the two-stage intersection



5-1 (b) Signal stage sequence of the two-stage intersection

Source: OSCADY PRO v1.1

Fig. 5-1 A two-stage traffic intersection and the movement of traffic in signal stages

The approximation function using (5-1) and (5-2) takes the form of

$$\tilde{J}(\cdot, r) = r'\phi(i)\,. \tag{5-3}$$

This indicates that parameter r^- is assigned to link in green signal and r^+ to link in red signal. The two parameters are updated simultaneously by using Eq. (3-81) and (3-82), except that updates are suspended during inter-green periods. Other specifications of the ADP_PL mode remain the same as described in Section 4.9.1. Overall, this approach is a simplification of the general formulae presented in Section 4.9.1.

We specify the control policy for the two-stage intersection in the next section.

5.2.1.1 Control policy

At 5s resolution, according to Assumption 4-2, the inter-green and minimum-green take one temporal interval each. According to Assumption 4-3, details of vehicle behaviour within the interval are neglected. According to Assumption 4-6, data of 10s future vehicle arrivals becomes available at the beginning of the interval.

We further set the planning period as 10s only, which means that there is no 'tail' part (explained in Section 4.7.2) in this case. This also means that there are only two opportunities for action during the planning horizon, one at the current time t and one at time $t + \Delta t$, where $\Delta t = 5$s. Since those settings and assumptions are identical to DYPIC (Robertson and Bretherton, 1974), we adopt the near optimum control policy proposed in their study. The near optimum policy has the following action space

(c) The signals are not changed, or

(d) The signals change immediately, or

(e) The signals are planned to change in 5s time.

The signals are changed only if decision (b) gives less total delay than both decision (a) and (c).

An example of making decision $\mathbf{u}_t$ of changing signals from green in Link B to green in Link A is

$$\mathbf{u}_t = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

where in order to maintain an inter-green after the end of green, the signal of Link B is immediate switched to red, whilst the red signal in Link A remains until the inter-green expires. Similarly, in order to implement option (a), we have

$$\mathbf{u}_t = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Using the rolling horizon approach, only the first part of $\mathbf{u}_t$ is implemented, and then the system rolls forward. However, implentation of (b) means that the controller has to wait for two time increments before the next action becomes admissible. This is because of the mandatory inter-green and minimum green periods.

5.2.1.2 Stepsize rule and discount rate

Using ADP_PL system, the parameters of the approximation function are updated through Eq. (3-82):

$$r_{t+1} = (1-\eta_t)r_t + \eta_t \Delta r_t,$$

where the stepsize η_t scales the new estimation to correct the current estimation. In the two-stage case, we use a deterministic stepsize rule

$$\eta_k = \frac{1}{k}, \tag{5-4}$$

where k denotes the number of updates up to the current moment. It is easy to show that (5-4) produces an estimate r_t that is an average of all previous observations, i.e.,

$$r_t = \frac{1}{k}\sum_{m=1}^{k} \Delta r_m . \tag{5-5}$$

In general as $k \to \infty$, parameter r_t converges to the mean value. Powell (2007, Section 6.5.1) shows that this stepsize rule is optimal when Δr_m is a sequence of unbiased estimations, the underlying data is stationary and we have no priory information of the sample mean.

Although this deterministic stepsize rule fits well with adaptive estimation and is easy to implement, a general concern is that the stepsize declines to zero so fast that the estimate may not converge to the true mean. A typical situation in experiment is that the system experiences a transient state in the early stage of simulation, during which the estimate Δr_m is biased, for that Δr_m is dependent on $\tilde{J}(\cdot, r)$. Using the $1/k$ rule, the system assigns the higher weights to estimations in transiant state where accuracy is poor. Notwithstanding this, we use this stepsize rule here for its simplicity.

The value for discount factor α of the ADP algorithm has to be decided manually. Let the discount factor α be defined as

$$\alpha^t = \exp(-\theta t), \tag{5-6}$$

where $t = k\Delta t$, $k = 1, 2, \ldots, K$. We find the optimal θ for the ADP_PL algorithm through pilot experiments. At Δt = 5s, we found that $\theta = 0.05$ per time increment was the favourable value. This is equivalent to say that future vehicle delay is discounted at 1% per second.

5.2.1.3 Control strategies for benchmarks

The DYPIC method is a backward dynamic programming (BDP) approach to solve a simple case of traffic signal control. In this study we also use the results from the BDP approach as the benchmark on the higher bound (global optimum). It is worth recalling that for adopting BDP solution approach, the complete information for the entire operation period is required, and the calculations are costly. Robertson and Bretherton also found that their near optimum policy of planning 10s ahead and roling forward 5s causes about one second more delay per vehicle than DYPIC in a range of random and cyclic traffic flows. As discussed in Chapter 2, this near optimum policy is an early attempt of employing the principle of ADP. We use this near optimum policy as a method for comparison, and denote it as the RB method.

The third method for comparison is Gartner's (1982) optimum sequential constrained algorithm (OSCO), which is used for OPAC. This approach is not derived from dynamic programming but from exhaustive search algorithm.

Fixed-time strategy is not included here. This is because all the methods for comparison quoted above performed significantly better than the optimal fixed-time timings in various experiments, including both computer simulation and field test. We are more interested here the performance of ADP in respect to BDP, RB and OSCO.

5.2.1.4 Performance

Results of included control strategies are grouped into Table 5-2 and Table 5-3. The former table summarises the averaged performance of a single simulation run of 120 temporal increments (Δt =5s), whilst the latter summarises the averaged performance over 7200 temporal increments. The performance of ADP in the short-run, in comparison with other methods, varies among experiments. When traffic flows are light, as shown in the left columns of Table 5-2, ADP performed better than RB and OSCO in general. With higher traffic flows, the performance of ADP fluctuates around that of RB, but is consistently better than OSCO. Two factors may contribute to this occurrence: first, the initial values of r ; second, the effects of transition state.

Table 5-3 shows that the performance of ADP_PL in the long-run is as good as RB, and consistently better than OSCO. Another noticeable feature of ADP_PL is that it can operate with both under-saturated and over-saturated traffic, whilst the RB approach is applicable only for under-saturation.

Table 5-2 Averaged vehicle delay (vehicle seconds per second) of simulations of 120 temporal increments; two-stage intersection at 5s resolution

Link A / Link B	Method	252 V/h	396 V/h	600 V/h	678 V/h
240 V/h	BDP	0.52	0.73	1.34	1.67
	RB	0.75	1.24	1.63	2.09
	OSCO	0.71	0.91	2.31	2.51
	ADP	0.70	0.93	2.08	2.40
432 V/h	BDP	1.02	1.72	3.48	4.05
	RB	1.31	2.05	4.01	4.51
	OSCO	1.42	2.33	4.14	5.38
	ADP	1.30	1.96	3.88	4.53

Table 5-3 Averaged vehicle delay (vehicle seconds per seconds) of simulations time period of 7200 temporal increments, two-stage intersection at 5s resolution

Link A / Link B	Method	252 V/h	396 V/h	600 V/h	678 V/h
240 V/h	BDP	0.51	0.88	1.52	1.86
	RB	0.60	1.02	1.78	2.16
	OSCO	0.65	1.14	1.90	2.31
	ADP	0.54	0.94	1.81	2.27
432 V/h	BDP	0.99	1.73	3.36	4.38
	RB	1.19	1.97	3.51	4.46
	OSCO	1.21	2.10	3.78	4.91
	ADP	1.13	1.97	3.52	4.46

Considering that the RB method uses a second order polynomial approximation function, the performance of ADP_PL with linear approximation function shows that provided the monotonicity of the value function is preserved, and the approximation updated properly, there is no significant difference in performance between linear and non-linear approximation. Given the simplicity of linear functions, and the online learning techniques available for adjusting their parameters, the result encourages us to use linear function for further investigations for more complex case studies.

BDP is used here to benchmark the lower bound in vehicle delays. Since BDP solution requires the complete knowledge of exogenous vehicle arriving process for the entire simulated time, we generated the complete vehicle arrivals for 7200 temporal increments and stored them in information profiles. The BDP calculation started from the last time interval, and used the information profiles for each iteration of computing. However, such availability of information is unrealistic in real-time control.

The performance gap between ADP and BDP reduces in the long-run, and the short-run results may well be influenced by the transient state of computer simulation. In Fig.5-2 we compare the performance of the two strategies by plotting inflow and outflow profiles and the evolution of queues on Link A. This comparison consists of two parts. The first compares the performances in the first 10 minutes, thus including the transition state. The second part compares the performance in the last 10 minutes, which includes the steady-state.

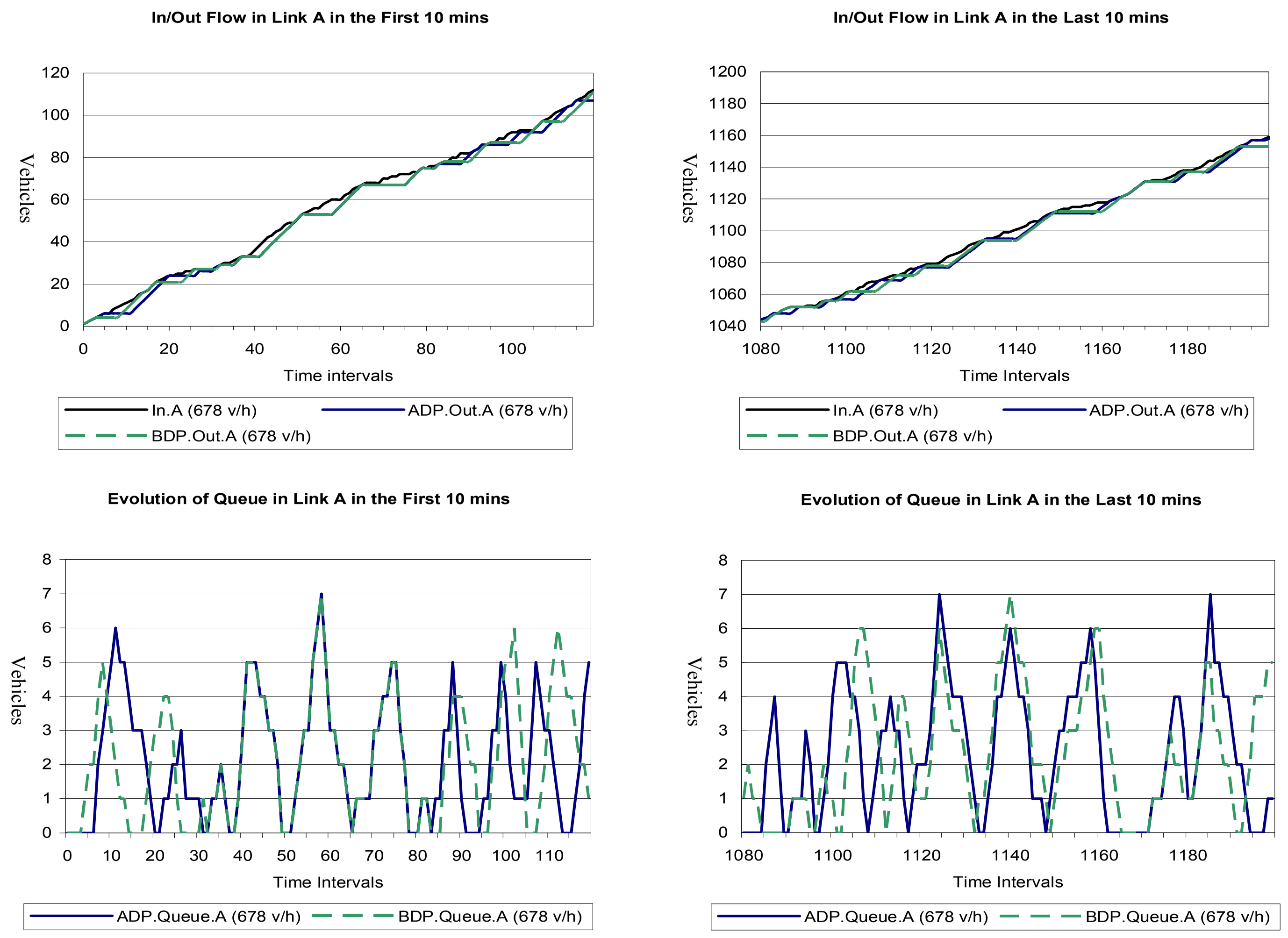


Fig. 5-2 A comparison between ADP_PL and BDP in In/Out flows and the evolutions of queue length on Link A

In both cases, ADP and BDP produced the same number of cycles so the difference in performance is due to the green splits and variations in the cycle lengths. When queues are long in the system and traffic is heavy, e.g. during the time period between intervals 40 and 80, and then between 1120 and 1160, the two methods produced similar or even identical signal plans. This can be explained by that, when the intersection is congested, the optimal solution is to dissipate queues on green link until it is empty, or as much as possible. Vehicle arriving in the future becomes less influential to decision-making. On the other hand, when queues are rare and traffic is light, BDP performs better than ADP. This is because under such condition, a quick response to future arrivals is essential to reduce delay. However, the BDP approach requires complete information to achieve the global optimum, while the ADP approach uses arrival information for next 10 seconds. Furthermore, the BDP takes about 10 minutes to complete a single simulation run of 7200 time increments, whilst the ADP takes about 5 seconds, which is about 100 times faster to compute.

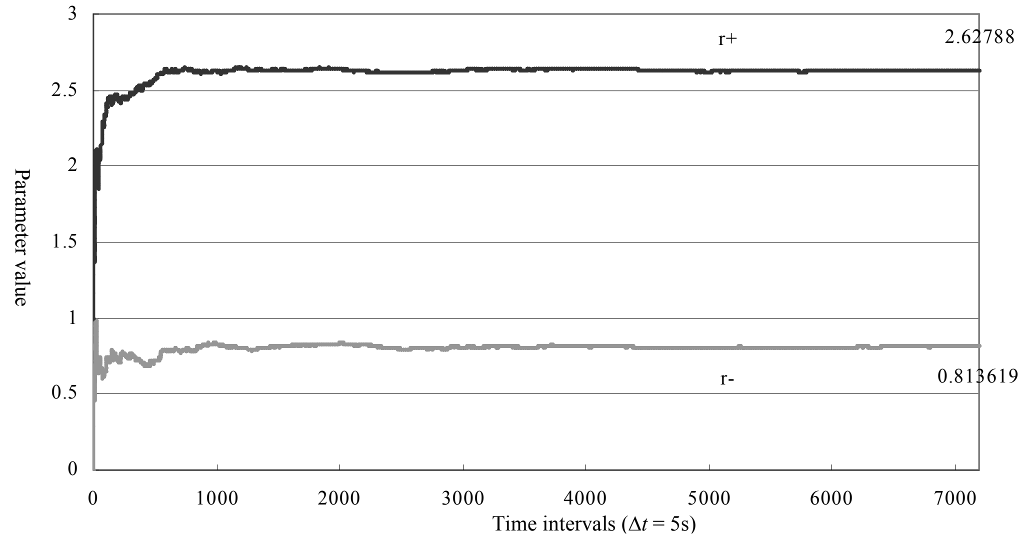
5.2.1.5 Parameters of the approximation function

The values of functional parameters appear to have stabilised in simulation. Shown in Table 5-4, values of r^+ and r^- (of Link A) are proportionate to the degree of saturation, which means that the higher the degree of saturation, the more additional delay per vehicle. In each case, the values of r^+ are greater than those of r^-, indicating that vehicles in red link experience more delay than vehicles in green link do. Those tables show the monotonicity of the value function with respect to degree of saturation.

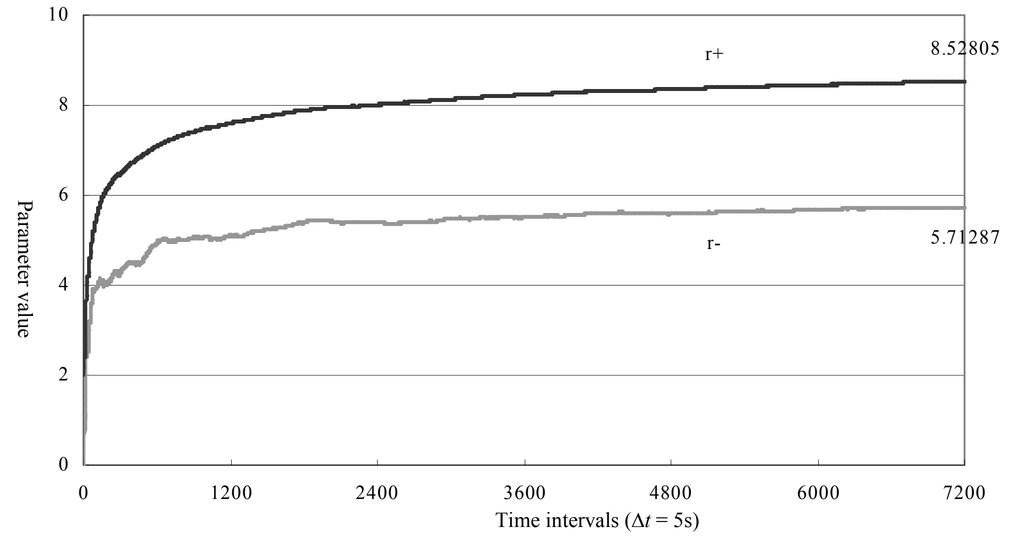
Table 5-4 Terminal values of r^+ and r^-, two-stage traffic intersection at 5s resolution

Link A / Link B	Coefficient	252 V/h	396 V/h	600 V/h	678 V/h
240 V/h	r^-	0.814	1.468	2.416	2.818
	r^+	2.628	3.602	5.046	5.648
	$\|r^- - r^+\|$	1.814	2.134	2.63	2.83
432 V/h	r^-	1.629	2.578	4.626	5.713
	r^+	3.82	5.104	7.526	8.528
	$\|r^- - r^+\|$	2.191	2.526	2.9	2.815

We take two examples from the experiments to illustrate graphically the evolution of the parameters. The first example is shown in Fig. 5-3 (a), with 396 v/h in Link A and 240 v/h in Link B. This is a relatively light flow condition. Shown in Fig. 5-3 (b), the second example represents a heavier flow condition, with 678 v/h in Link A and 432 v/h in Link B. The values appear to have converged after $t = 1000$ in the lighter flow condition. In the heavier flow condition, however, the indications of convergence are weak.



5-3 (a) Evolutions of r^+ and r^- (396 v/h in Link A and 240 v/h in Link B), two-stage intersection controlled by ADP_PL



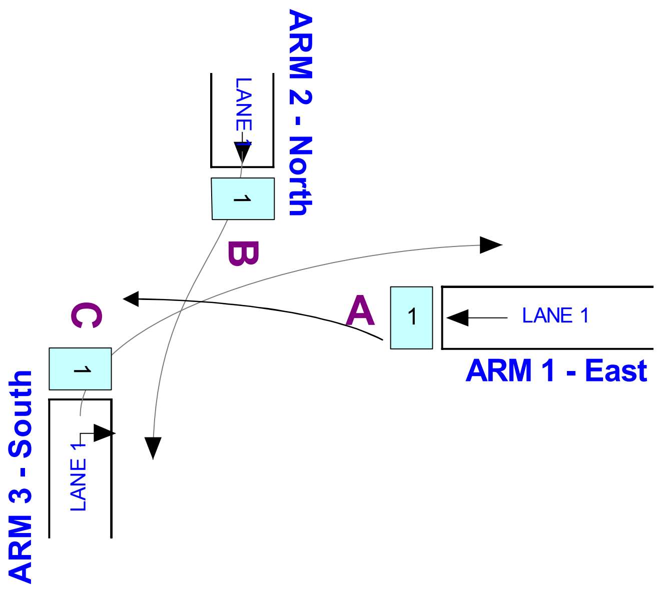
5-3 (b) Evolutions of r^+ and r^- (678 v/h in Link A and 432 v/h in Link B), two-stage intersection controlled by ADP_PL

Fig. 5-3 Evolutions of parameters of the approximation function in the case of two-stage isolated intersection

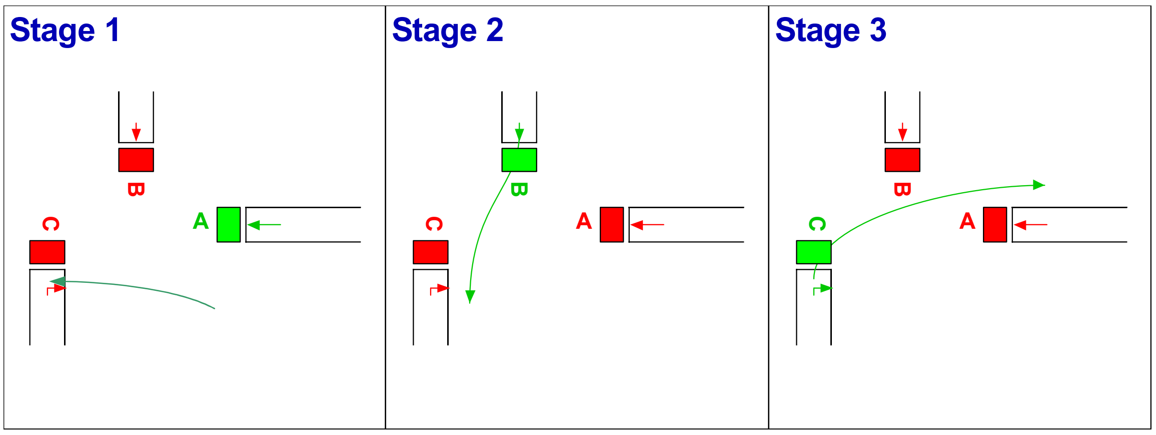
These results show that the ADP_PL controller is sufficient for real-time operation at a simple isolated intersection. In the following experiment, we investigate the performance of the ADP controller in the cases of multiple-stage intersection.

5.2.2 Three-stage intersection at 5s resolution

Numerical results from experiment on a three-stage intersection with ADP controller were first presented in Heydecker *et al* (2007). The discussion is expanded in this section.



5-4 (a) Geometric layout of the three-stage traffic intersection



5-4 (b) Signal stage sequence of the three-stage traffic intersection
Source: OSCADY PRO v1.1

Fig. 5-4 Signal stage sequence and traffic movements for the three-stage traffic intersection

The layout of the three-stage intersection is shown in Fig. 5-4 (a). Traffic links A, B, C have green during stages 1, 2, 3 respectively, and each link has one approach lane. Traffic in Link A travels straight ahead from east to west, and traffic in Link B from north to south. Traffic in Link C turns right from south to east. The sequence of signal stages in the fixed-time plans is shown in Fig. 5-4 (b). With this configuration, we are able to demonstrate the acyclic signal timings of the ADP controller to compare the results with optimised cyclic signal timings.

Both 5s and 0.5s resolutions are investigated in this case. We begin the experiment on ADP_PL at 5s resolution, which is a direct extension of the two-stage case, and compare the results with the optimised fixed-time plans. In the three-stage case, we do not consider other benchmarking strategies, such as OSCO and RB. This is because most of the literature on adaptive traffic signal control only describes the formulae concerning two signal stages. Extending the formulae to multi-stage case may require inventions that deviate from true intention of related control methods.

Since this case is broadly an extension of the experiment on the two-stage case, all of the assumptions and traffic regulations are kept the same as for the two-stage case, except for the additional traffic link and signal stage. The total traffic demand at the intersection is 1116 v/h, with the junction capacity at 1440 v/h. The distribution of traffic demand among each approach link is shown in Table 5-5, together with the degrees of saturation for each signal stage and the intersection. We introduce the following notations[2] for the calculation of degrees of saturation:

q mean arrival rate

s saturation departure rate (or saturation flow)

y flow ratio: $y = q / s$

λ green proportion

x degree of saturation: $x = y / \lambda = qc / gs$

[2] The notations are conventional in transport study and only apply for the case presented in Table 5-5.

Table 5-5 Traffic demand and degrees of saturation for the three-stage intersection obtained from Webster's method: fixed-cycle length c = 125s, total lost time L = 15s and equal saturations among signal stages

Traffic links	Arrival q_i (v/h)	Saturation flow s_i (v/h)	Flow Ratio $y_i = q_i/s_i$	Green proportion $\lambda_i = y_i\,(c\text{-}L)/c\sum y_i$	Degree of saturation $x_i = q_i/\lambda_i\, s_i$
Link A	432	1440	0.30	0.34	0.88
Link B	252	1440	0.18	0.20	0.88
Link C	432	1440	0.30	0.34	0.88
Sum	1116	-	0.78	0.88	-

For adaptive control, cycle length and green proportion are variables, and calculating mean cycle length is complicated for a multi-stage adaptive problem. To provide a basic estimation of x, we obtain optimised fixed-time cycle length using Webster's formula (1957):

$$c = \frac{1.5L + 5}{1 - \sum_i y_i}$$

Assuming that L = 15s (3×5s inter-green) and s_i = 1440 v/h for link i, we have c = 125s. Effective green split λ_i is calculated from

$$\lambda_i = \frac{y_i\left(c - L\right)}{c\sum_i y_i}.$$

As shown in Table 5-5, the degrees of saturation reach 88% of the capacity, indicating a heave but still under-saturated traffic situation.

Again, the degrees of saturation shown in the table are for reference only. In adaptive control, degrees of saturation are time-variant as cycle length and green splits are adaptive in response to changes in traffic demand.

5.2.2.1 Control policy

The control policy is extended from the two-stage case to include the decision sets below

(a) The signals are not changed

(b) The signals change immediately to the signal stage that gives least total delay

(c) The signals change after Δt to the signal stage that gives least total delay.

The signals are changed only if decision (b) gives less total delay than both decision (a) and (c). We only consider a planning horizon of 10s. Consequently, the planning horizon only has the head part, which is supplied with detected data. Using rolling horizon approach, only the first 5 seconds of the signal plan is implemented, and then the system rolls forward.

5.2.2.2 Performance and comparison

For the ADP_PL algorithm, we continue to use the 1/*k* stepsize rule. The initial values for the approximation function parameters are set as $r^- = 1$ and $r^+ = 2$ for all links. The initial system state is queue-free, with green signal for Link A.

Before conducting a full-scale experiment, we use pilot experiments to find a favourable value for discount factor *θ*. The results of the pilot experiments are shown in Table 5-6. Each pilot test has a simulated time of 3 hours and 20 minutes. All the pilot experiments use identical simulation input. Shown in Table 5-6, the optimal value for *θ* is 0.8% per second.

Table 5-6 Discount rate *θ* and performance in pilot tests

Discount rate *θ* (per second)	Performance (v.s/s) with 1/*k* stepsize
0.2%	8.31
0.4%	8.31
0.6%	8.34
0.8%	8.23
1.0%	9.16

Table 5-7 Optimised fixed-time plan for three-stage intersection

Stage Sequence (left to right) of a signal cycle						Cycle length
Inter-green	Stage 2	Inter-green	Stage 3	Inter-green	Stage 1	
5s	25s	5s	40s	5s	40s	120s

Table 5-8 Averaged delays (v.s/s) and standard deviations of 20 runs, three-stage intersection at 5s resolution

	ADP_PL	Fixed-time
Average (v.s/s)	8.64	16.62
% of Fixed-time	52%	100%
Standard deviation between runs (v.s/s)	0.58	1.10

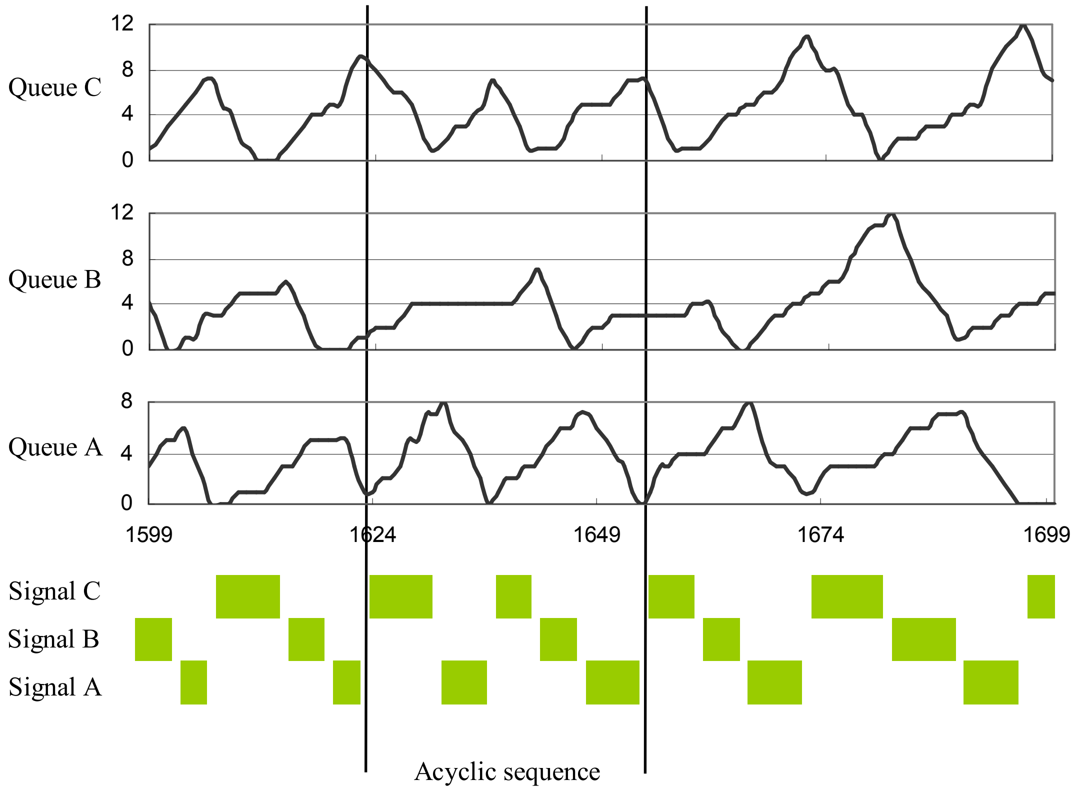


Fig. 5-5 Signal stage sequence and queue evolutions, ADP_PL at 5s resolution; X-axis indicates time units, Y-axis indicates queue length in vehicle units

Stage sequence and green splits of the optimised fixed-time plan are shown in Table 5-7. The optimised fixed-time plan is found by exhaustive search in computer simulation rather than using analytical methodologies. The maximum cycle time is set at 120 seconds, a common constraint in signal practice in the U.K (DoT, 1981). We then conduct 20 independent simulations, with each of 2400 time increments. The averaged results of ADP_PL and fixed-time plans are summarised in Table 5-8. The ADP_PL controller reduces 48% vehicle delays from the best fixed-time plans.

5.2.2.3 Stage sequence and green splits

One of the advantages of the ADP controller comes from the ability to adjust signal timings according to the dynamics of the traffic without referring to preset control plans. An example of such acyclic sequence and variable green splits produced by ADP_PL controller is shown in Fig. 5-5. During the period between $t = 1599$ to $t = 1699$ ($\Delta t =$ 5s), the predominant stage sequence is A-C-B-A. However, between $t = 1624$ to $t = 1649$, the controller skipped Stage 2 once, and generated a sequence A-C-A-C-B-A. This sequence coincided with the queuing dynamics that resulted from the absence of arrivals on Link B between $t = 1624$ to $t = 1649$. When the queue in Link C was cleared, the controller switched green signal to Link A, whose queue and arriving traffic are greater than those of Link B. A distinct feature demonstrated by Fig. 5-5 is that the controller normally switches signal indication when the queue in the current green link is either empty or nearly dissipated.

5.2.2.4 Parameters of the approximation function

The evolution of the parameters are shown in Fig. 5-6, together with moving average of the vehicle arrival rates and vehicle delays.

The cumulative moving average of vehicle arrivals shown in Fig. 5-6 (a) demonstrates two distinct states in computer simulation: the time period up to $t = 2400$ which is regarded as transient-state, and the time period afterwards which is regarded as steady-state. It is worth noting that there are more systematic ways to determine the actual boundary between the transient-state (or the warm-up period) and steady-state in computer simulation, such as the Welch method (1981, 1983). The Welch method, which smoothes out high-frequency oscillations in estimation by averaging corresponding observations over several replications, still relies on graphical procedures to estimate the boundary beyond which estimation appears to have converged. Given the apparent patterns in our graphical display in Fig. 5-6 (a), we consider that the system enters steady-state after t =2400.

During the transient-state, the vehicle arriving rate steadily rises to the designated statistical mean at 1116 v/h (see Fig. 5-6(a)), despite the random fluctuation. The cumulative moving average for mean rate of delay also rises steadily throughout the same period, as shown in Fig. 5-6 (b). The vehicle arriving rate broadly stabilises with the actually value rises slightly from 1116 v/s to 1130 v/s. Consequently there is also a slight increase in mean rate of delay from 8.87 v.s/s to 9.19 v.s/s.

The parameters of the approximation function rise rapidly in the transient-state and then become stable in steady-state. Parameters r^+ are consistently greater than r^-, and the differences between them are broadly constant across different links in steady-state. Although the designated traffic flow in Link B is only 58% of Link A and Link C, the terminal values of r^+ (10.98) and r^- (7.96) are greater than their counterparts of Link A and C. This indicates that vehicles approaching on Link C experience more individual delays than those on other links. This effect balances the degree of saturations among the approaching links. Although we used the stepsize rule of $1/k$, whose value rapidly decreases toward zero, parameters are not converged after 7200 increments.

In the next section, we investigate performance of ADP controllers at 0.5s resolution.

5.2.3 Three-stage intersection at 0.5s resolution

Studies on multiple-stage intersection control using ADP at fine resolution were first presented in Cai *et al.* (2009). At the 0.5s resolution, traffic arrivals are generated by the shifted Bernoulli process described by Eq. (4-1) and (4-2), which prevent vehicles from arriving in quick succession. We recall that the performance measured under this fine resolution is not directly comparable with performance measured at coarser resolutions. As we have described in Section 4.2, in the coarser case (5s), no delay is attributed to either vehicle in a time increment during which two depart, whilst in the finer case, delay would occur to one of the vehicles for the whole of the departure time of the other. We can only compare the control performance by transferring the signal plans from the coarse resolution to the fine.

All assumptions and regulations remain the same as in the coarser case, unless wherever specified. Both ADP_PL and ADP_TD algorithms are investigated in this case. As we have discussed in Section 4.9.1, the two algorithms differ only in the learning technique that is adopted. The ADP_PL algorithm has demonstrated stability in previous cases and produced competitive results. Introducing ADP_TD in this case extends our investigation in ADP, and offer comparisons between optional learning techniques.

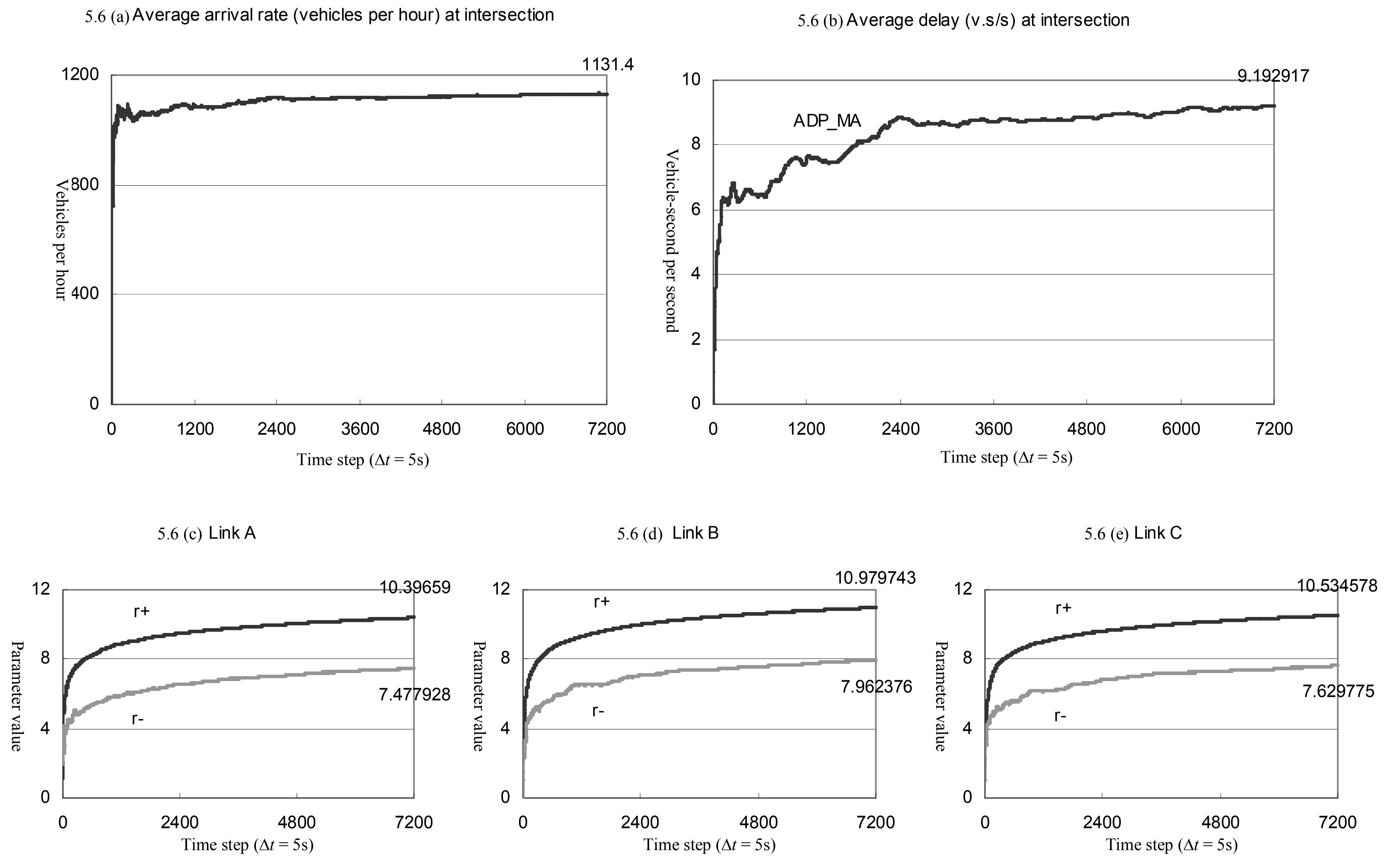


Fig. 5-6 (a) and (b) show the cumulative moving average of vehicle arrival rate and delay in a time period of 7200 time increment (X-axis); (c), (d) and (e) show the evolution of approximation function parameters; controller type is ADP_PL at 5s resolution

We establish two scenarios for investigations in this case. The first scenario uses a stationary traffic inflow the same as specified in Table 5-5. The second scenario uses a traffic inflow profile consisting of pre-peak, peak and post-peak arrival rates. We are especially interested in the evolution of function parameters in a non-stationary traffic environment.

The control policy is modified to consider the higher frequency of revising signal plans.

5.2.3.1 Control Policy

In this case, the ADP controller considers the following possible 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 after $k\Delta t$ seconds, for $k = 1, 2, \ldots, T_h - 1$, to the signal stage that gives least total delay, where T_h is the total number of temporal increments in the head part of the planning period.

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

The evaluation of optional decision (c) requires a tail part of planning period. This is because we want the planning period to be long enough to avoid ending horizon in inter-green, where all signals are red. For example, in evaluating option (c), the decision of signal change after time $k = T_h - 1$ is followed by a mandatory inter-green of 5s. This means that if the total number of steps $M = T_h$ (planning horizon equals to head period), the terminal signal state at $k = T_h$ will be all-red, as the inter-green state defines. Given a planning period of M-step, the possibility of terminating at all-red state will make option (c) unfavourable because all other options terminate in states with one traffic stage in green.

With regard to the duration of the tail part of the planning horizon, we prefer to keep it at 10s or less. The reasons for this are that, first, we have shown in the preceding tests that planning for 10s at coarse resolution is efficient and effective; second, a longer tail part may shift the weight from head to tail, thus from detected information to predicted information.

With the tail part being 10s, we have a horizon of 20 seconds, equivalent to 40 temporal increments at the finer resolution.

An example of making decision $\mathbf{u}_t$ of immediate change from green in Link A to green in Link C is shown as the following:

$$\mathbf{u}_t = \left[\underbrace{\begin{matrix} \overbrace{\begin{matrix}0 & 0 & \cdots & 0\end{matrix}}^{\text{Inter-green}} & \overbrace{\begin{matrix}1 & 0 & \cdots & 0\end{matrix}}^{\text{Minimum-green}} & \begin{matrix}0 & 0 & \cdots & 0\end{matrix} \\ \begin{matrix}0 & 0 & \cdots & 0\end{matrix} & \begin{matrix}0 & 0 & \cdots & 0\end{matrix} & \begin{matrix}0 & 0 & \cdots & 0\end{matrix} \\ \begin{matrix}\vdots & \vdots & & \vdots\end{matrix} & \begin{matrix}\vdots & \vdots & & \vdots\end{matrix} & \begin{matrix}\vdots & \vdots & & \vdots\end{matrix} \\ \begin{matrix}1 & 0 & \cdots & 0\end{matrix} & \begin{matrix}0 & 0 & \cdots & 0\end{matrix} & \begin{matrix}0 & 0 & \cdots & 0\end{matrix} \end{matrix}}_{\text{40 steps}}\right].$$

The prediction of the traffic information for the tail part is generated by Monte Carlo simulation. Using detected information, the controller records a moving average of traffic rate for each traffic link, which is then used as input to Eq. (4-2). The reminder of the process uses the inverse transformation method (ITM) as described in Section 4.2. This approach conforms to the definition of fixed tail (Section 4.7.2), which is suitable for online implementation since it only requires data that are readily available from detectors.

5.2.3.2 Stationary Traffic

Our primary objectives in this scenario are to investigate:

1) benefits from operating at finer resolution;

2) whether the parameters of the approximation function evolve differently if updated by different learning techniques;

3) whether the different learning techniques result in difference in performance;

4) gap from the global optimum (BDP result) in performance measure.

For the ADP_PL approach, the parameters of the approximation function are updated every 0.5s. To avoid "apparent convergence," which means that the solution is far from the best that can be obtained, we use *generalised harmonic stepsizes* (George and Powell, 2006; Powell, 2007):

$$\eta_k = \frac{a}{a+k-1}. \tag{5-7}$$

This rule satisfies Assumption 3-6 (a), and produces larger stepsizes for $a > 1$ than the $1/k$ rule. Increasing a slows the rate at which the stepsize approaches to zero. Using results from pilot tests, we set $a = 40$ for ADP_PL.

The value of the parameter θ for discount function Eq. (5-6) is also determined through pilot tests, results from which are summarised in Table 5-9. We choose $\theta = 24$ % per second for further investigations in this test scenario, as this value gives good performance for both learning techniques. Using such a substantial discount rate means that future cost after some time becomes irrelevant (or almost so) to current decision-making, thus making the problem solving more myopic. This myopic feature becomes favourable in systems of fine resolution because what matters almost of all is the current state and the state of immediate future (as obtained from Bellman's equation), and there are plenty of opportunities to revise the decision to accommodate future state. In similar logic of reasoning, for systems of coarse resolution, the controller has fewer opportunities to adjust to changes in state, e.g. revising at every 5s instead of 0.5s. It makes sense for the controller to plan longer ahead to compensate the rigidity in revising plans.

Additionally, from Table 5-9, we found that increasing discount rate from 10% per second to 24% per second only reduced vehicle delays by 0.1 v.s/s on average, and too myopic a system reverses the gains in performance. This implies that at the fine level of resolution, the rapid revision of signal plans itself matters most in providing the good performance. In a similar manner, we found a favourable learning rate $\eta = 0.001$ which applies to Eq. (3-74) and (3-82) for updating parameters of the approximation function.

Table 5-9 Pilot tests on parameter θ and performance (v.s/s), three-stage intersection at 0.5s resolution

Parameter θ (per second)	Performance	
	ADP_TD (v.s/s)	ADP_PL (v.s/s)
10 %	4.49	4.40
20 %	4.48	4.37
22 %	4.48	4.43
24 %	4.38	4.36
26 %	4.38	4.43
28 %	4.46	4.43
30 %	4.33	4.42
40 %	4.42	4.50

5.2.3.2.1 Performance and comparisons

We obtained results from 10 independent runs of simulations for ADP_PL and ADP_TD, the results of which are summarised in Table 5-10. In light of the similarity in results, we set a hypothesis of equal mean, and used paired-*t* test to test the hypothesis. We found that $| t | = 1.496$, which is less than the one-tail critical value at 95% degree of significance $t = 1.833$, hence accepting the hypothesis of equal mean. This shows that at the finer resolution, and by discounting future cost at 24% per second, the two learning techniques are similar in performance.

In order to compare results from different resolution, signals plans of ADP_PL at coarser resolution were recorded and implemented in the finer case, so that they could be evaluated on a comparable basis. As shown in Table 5-10, operating at the finer resolution reduces vehicle delays by 42% from the coarser case. Advantages of operating at finer resolution come from the ability to revise and adjust signal timings more frequently according to the detected information. We depict a sample comparison in Fig. 5-7, in which signal sequences and queuing dynamics between $t = 6300$ and $t = 6800$ are plotted for each of the finer case and the coarser case. With identical traffic arrivals, the controller made 12 signal switches during the 250-second period in the finer case, producing a signal sequence as the following:

Table 5-10 Performance comparisons between ADP_TD and ADP_PL for the case of three-stage intersection

	ADP_TD	ADP_PL	ADP_PL
Run	0.5s	0.5s	5.0s
1	4.38	4.36	7.51
2	4.69	4.67	9.10
3	5.03	5.09	8.62
4	4.27	4.34	7.40
5	4.63	4.74	7.81
6	5.15	5.20	8.83
7	4.05	4.02	6.68
8	4.45	4.35	7.32
9	5.11	5.19	8.36
10	4.46	4.61	7.83
Mean	4.62	4.66	7.94
SD	0.37	0.40	0.76

B – C – A – B – A – C – A – C – B – A – C – B – A,

On the other hand, the controller made 9 switches in the coarser case and produced the following signal sequence:

C – A – C – B – A – C – A – B – C – A.

Comparison shows that in the finer case vehicles in each link waited less and experienced less delay than they did in the coarser case. The ADP_PL controller produced acyclic signal sequence in both cases.

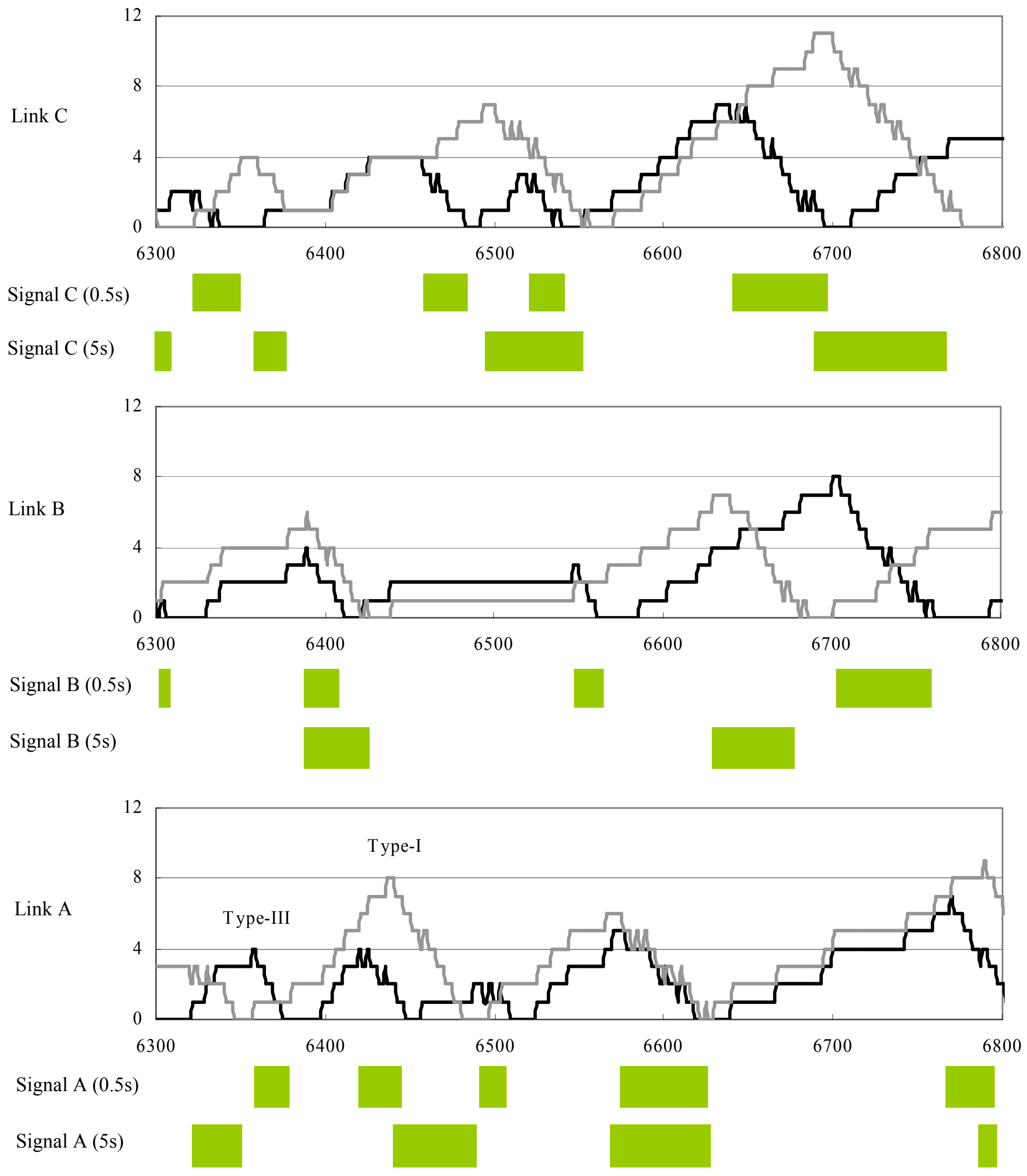


Fig. 5-7 Comparisons in signal sequences and queue evolutions produced by ADP_PL controller at 5.0s and 0.5s resolutions, three-stage traffic intersection; X-axis indicates time units, Y-axis indicates queue length in vehicle units

Table 5-11 Green time allocation (time increments) in a single simulation of 7200 time increments

Controller	ADP_TD	ADP_PL	ADP_PL
Resolution	0.5s	0.5s	5.0s
Link A	1899	1892	2120
Link B	1070	1132	1260
Link C	1941	1916	2130
Total	4910	4940	5510
% of *T*	0.68	0.69	0.77
No. switches	229	226	169

From another perspective, we compare green time allocation to links in Table 5-11. Green time allocations are broadly proportionate to traffic demand. Because the controllers made 57~60 more signal switches in the finer case, green time accounts for about 68~69 % of total time, comparing with 77% in the coarser case. This suggests that the ADP controllers in the finer case reduces about 42% of vehicle delay while using about 11% less green time than they do in the coarser case. What matters, therefore, it is the effectiveness of using capacity.

The upper bound performance (global optimum) is produced by BDP. In the finer case, the BDP approach is costly in computation. A single run of 720 time increments, equivalent to 6 minutes only, takes about 12 hours in a PC equipped with Pentium® Dual Core 3.60GHz and 3.50GB of RAM. For an operation period of 7200 time increments, the vehicle delay is 4.27 v·s/s from BDP approach in a single run, comparing to 4.62 v·s/s from ADP_TD and 4.66 v·s/s from ADP_PL over 10 runs.

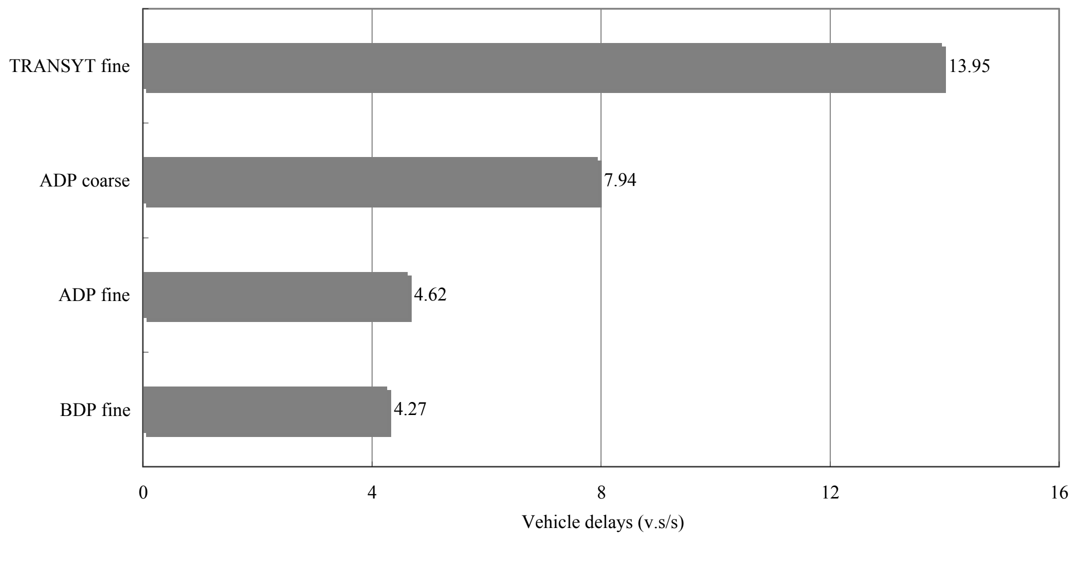


Fig. 5-8 Averaged control performance over 1 hour simulation for the case of three-stage intersection. TRANSYT, ADP and BDP are compared

Table 5-12 The optimised fixed-time plan from TRANSYT 12.0 for the case of three-stage 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 seconds

Number of Stages	Starting time			Cycle time
	Stage 1	Stage 2	Stage 3	
3	55	101	9	120 seconds

Table 5-13 Performance Comparison for the case of three-stage intersection. The decision sequences in coarse resolution are transferred to the fine resolution in order ensure consistence in comparison

	DP	**ADP_TD**	**ADP_PL**	**ADP_PL**	**TRANSYT**
Resolution (seconds per increment)	0.5	0.5	0.5	5.0	0.5
Simulation Time (minutes)	6	60	60	60	60
Run Time (minutes)	720	5.5	12	0.3	1/6
Discount Rate θ (per second)	24%	24%	24%	0.8%	–
Averaged Performance (v.s/s)	4.27	4.62	4.66	7.94	13.95

The lower bound of control performance is obtained from TRANSYT (Vincent *et al.*, 1980) version 12.0, the signal timings of which is summarised in Table 5-12. The optimised fixed-time plan resulted in 13.95 v.s/s at the 0.5s resolution. A full comparison in control performance of the aforementioned control methods is tabulated in Table 5-13. A graphical display of the differences in performance is shown in Fig. 5-8.

5.2.3.2.2 Parameters of the approximation function

The evolution of parameters under ADP_PL in a single run is plotted in Fig. 5-9, together with the cumulative moving averages of the arriving traffic rate and vehicle delays. The corresponding results of ADP_TD are plotted in Fig. 5-10.

Table 5-14 Mean and standard deviation of functional parameter in steady-state, ADP_PL at 0.5s resolution

	Link A		Link B		Link C	
	r^-	r^+	r^-	r^+	r^-	r^+
Initial value	1	2	1	2	1	2
Steady-state (2400 $\leq$ t < 7200)						
Mean	2.73	3.92	2.51	3.92	2.82	3.92
SD	0.38	0.00	0.47	0.00	0.34	0.00

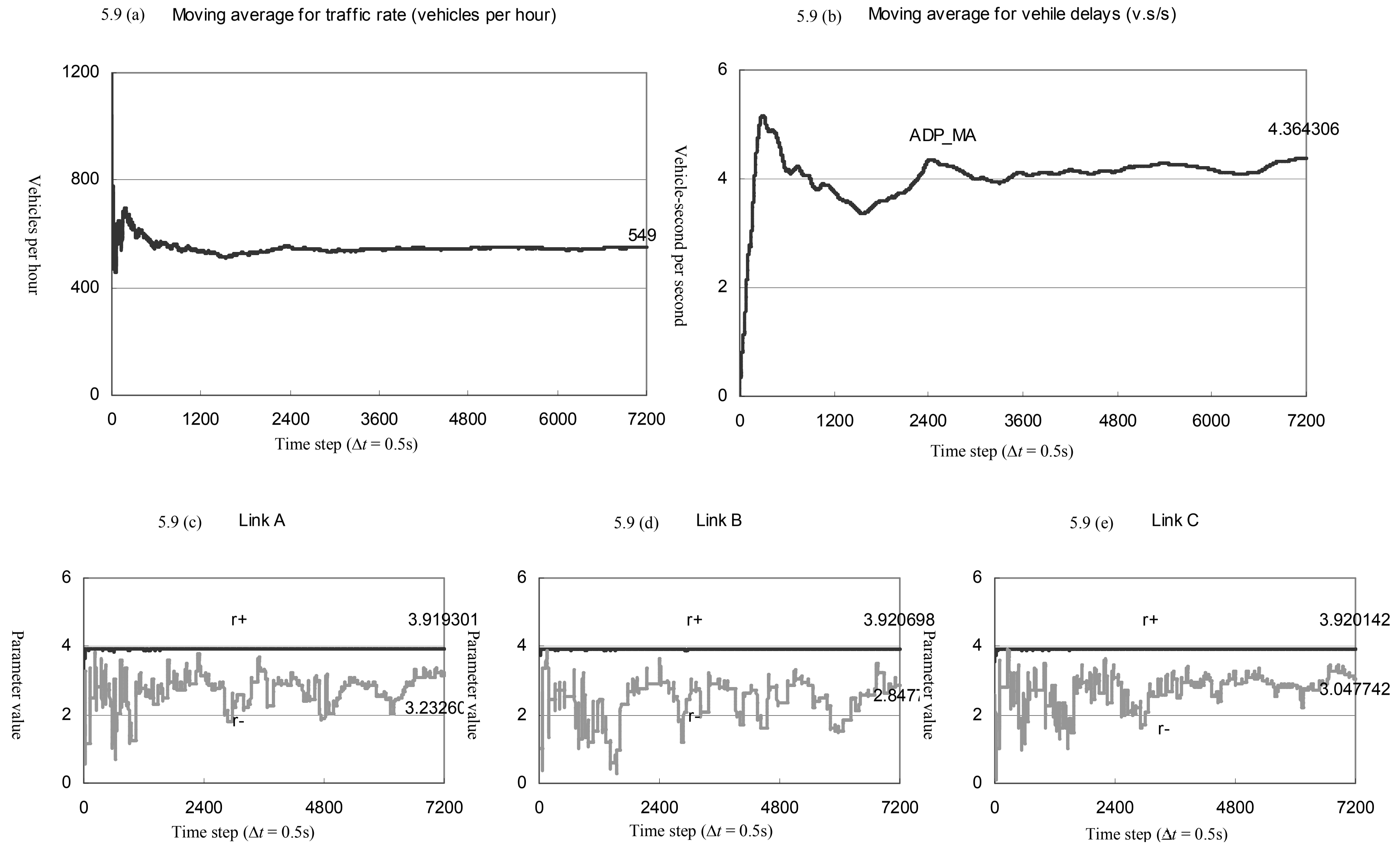


Fig. 5-9 Evolution of parameters of the approximation function, ADP_PL at 0.5s resolution. Figure (a) and (b) shows cumulative moving average of vehicle arrival rate and delay in a time period of 7200 time increment (X-axis); (c), (d) and (e) show evolutions of approximation function parameters.

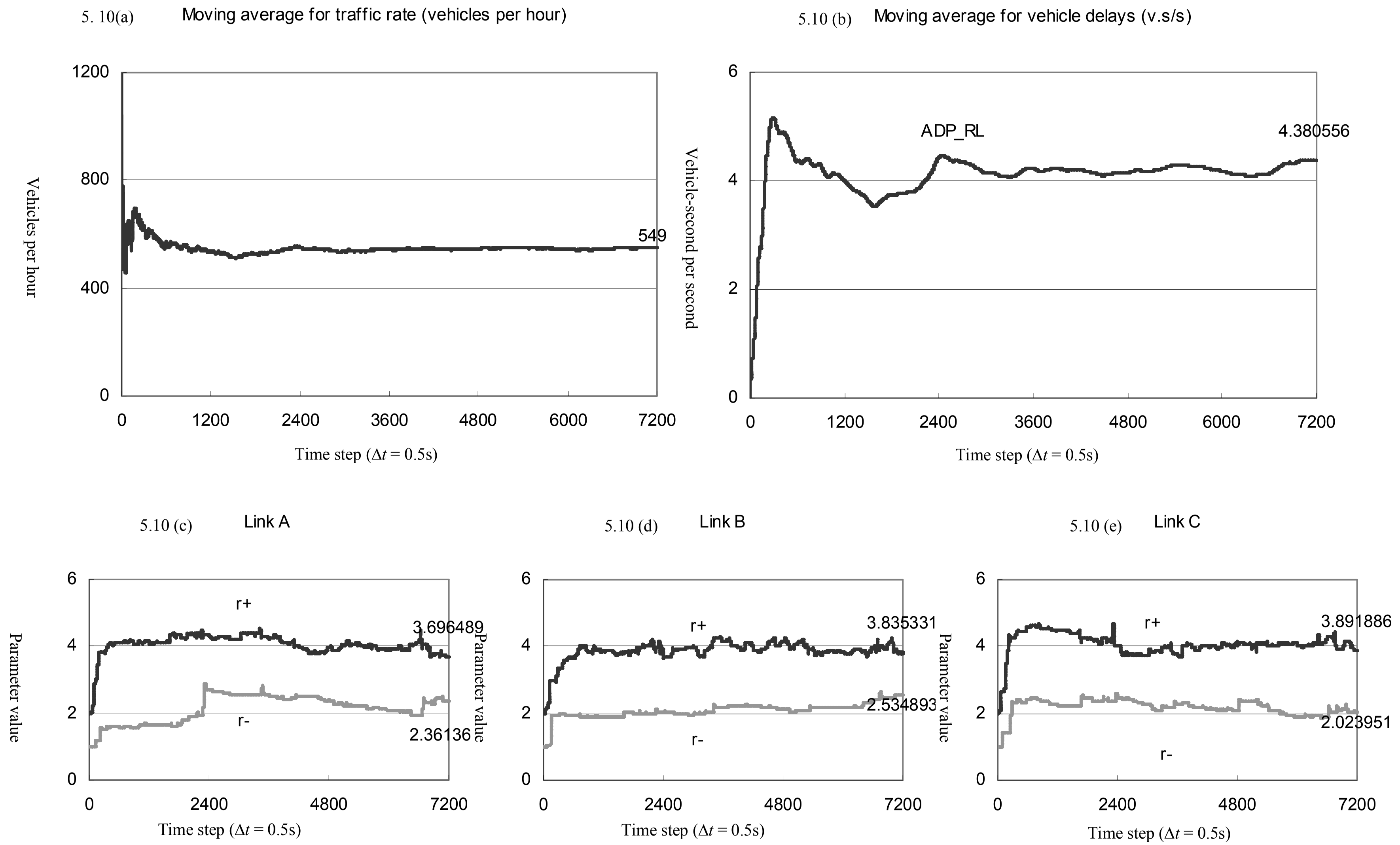


Fig. 5-10 Evolution of parameters of the approximation function, ADP_TD at 0.5s resolution. Figure (a) and (b) show cumulative moving average of vehicle arrival rate and delay in a time period of 7200 time increment (X-axis); (c), (d) and (e) show evolutions of approximation function parameters

Table 5-15 The convergence of functional parameters under ADP_TD

	Link A		Link B		Link C	
	r^-	r^+	r^-	r^+	r^-	r^+
Transient-state ($0 \leq t < 2400$)						
Mean	1.66	4.04	1.89	3.69	2.23	4.23
SD	0.30	0.47	0.23	0.45	0.34	0.53
Steady-state ($2400 \leq t < 7200$)						
Mean	2.36	4.06	2.18	3.95	2.15	3.99
SD	0.20	0.19	0.15	0.14	0.16	0.14

Fig. 5-10 (a) and 5-10 (b) show that the system under ADP_TD entered steady-state after $t = 2400$ as well. From Fig. 5-10 (c), (d) and (e), we find that the values of functional parameter show bounded fluctuation, which may result from the constant stepsize $\eta_t = 0.001$. We tabulate the mean value and standard deviation for each parameter in Table 5-15, and compare the statistics between transient-state and steady-state. The standard deviation of each parameter reduces substantially from transient-state to steady-state. Values of r^+ under ADP_TD are similar to those under ADP_PL, despite the different learning techniques and stepsize rules in use; parameters r^- under ADP_TD have smaller standard deviation than under ADP_PL in steady-state.

5.2.3.3 Non-stationary traffic

Our interest in this case is to investigate the response of the ADP controllers to changes in prevailing traffic, and the evolution of function parameters. The configuration of the traffic flow profile for each link is listed in Table 5-16. It consists of a pre-peak period, a peak period and a post-peak period. The transitions between these different traffic periods are smoothed, as shown in Fig. 5-11, with peak of traffic flow ocurrs at the middle point.

Table 5-16 Configuration of traffic flow profile

	Link A (v/h)	Link B (v/h)	Link C (v/h)	Total (v/h)
Pre-Peak	250	150	250	650
Peak	500	250	500	1250
Post-Peak	300	200	300	800

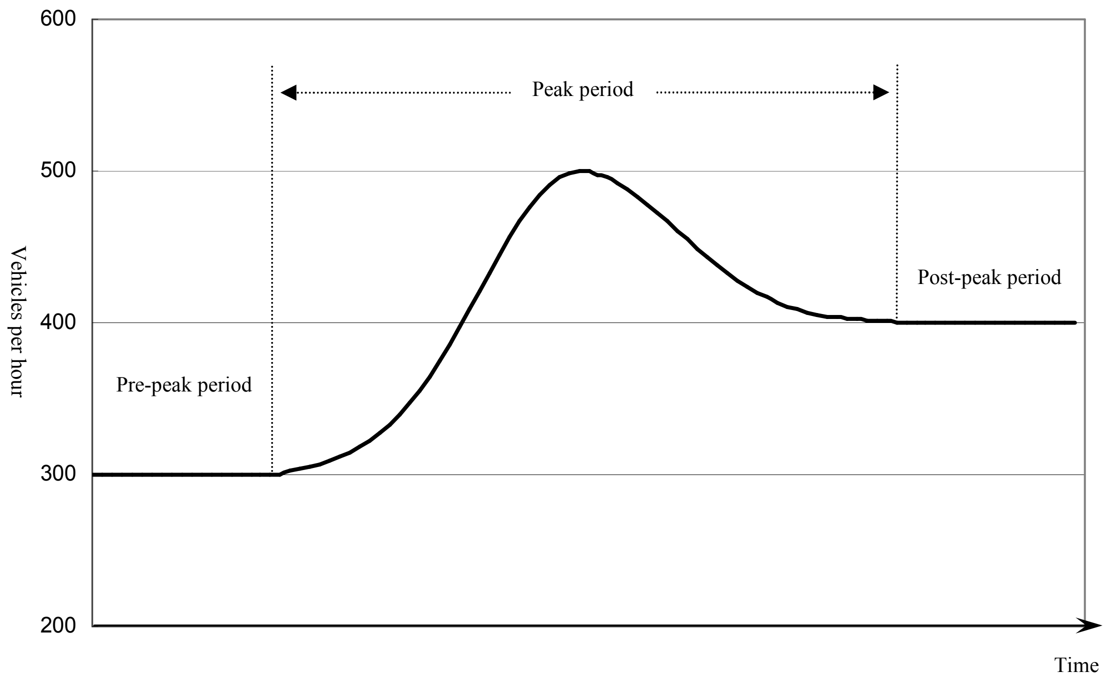


Fig. 5-11 A traffic flow profile for links A and C, containing pre-peak, peak, and post-peak periods; traffic rate in vehicles per hour

Both ADP_PL and ADP_TD are investigated in this case. The stepsize rules and discount rates are kept the same as in the stationary scenario. To allocate sufficient time for the system to develop to the steady-state, we simulated 28800 time increments at 0.5s resolution, equivalent to 4-hours of simulated time. Results of ADP_PL are plotted in Fig. 5-12, and those of ADP_TD in Fig. 5-13.

Shown in Fig. 5-12 (a) and 5-13 (a), the cumulative moving average of traffic flow rate experiences a pre-peak period, where the rate is broadly stable at 660 v/h (except for the transient-state), a peak at time t = 14400, where the slope of increase in traffic is the steepest, and a post-peak period, where the rate begins to decline. Consequently, as shown in 5-12 (b) and 5-13 (b), the cumulative moving average of vehicle delays reflects the same pattern as traffic flow rate. There is no discernible difference in performance between ADP_PL and ADP_TD.

The evolution of approximaiton function parameters exhibits similar patterns across links and between the two learning techniques. Shown in Fig. 5-12 (c), (d) and (e), values of r^+ under ADP_PL are insensive to the changes in traffic, whilst r^- is more reactive to changes in traffic. Using ADP_TD, and shown in Fig. 5-13 (c), (d) and (e), values of r^+ show in peak period a higher degree of variation, and are less stable than those under ADP_PL in general. Values of r^- rise sharply with substantial variation in peak period.

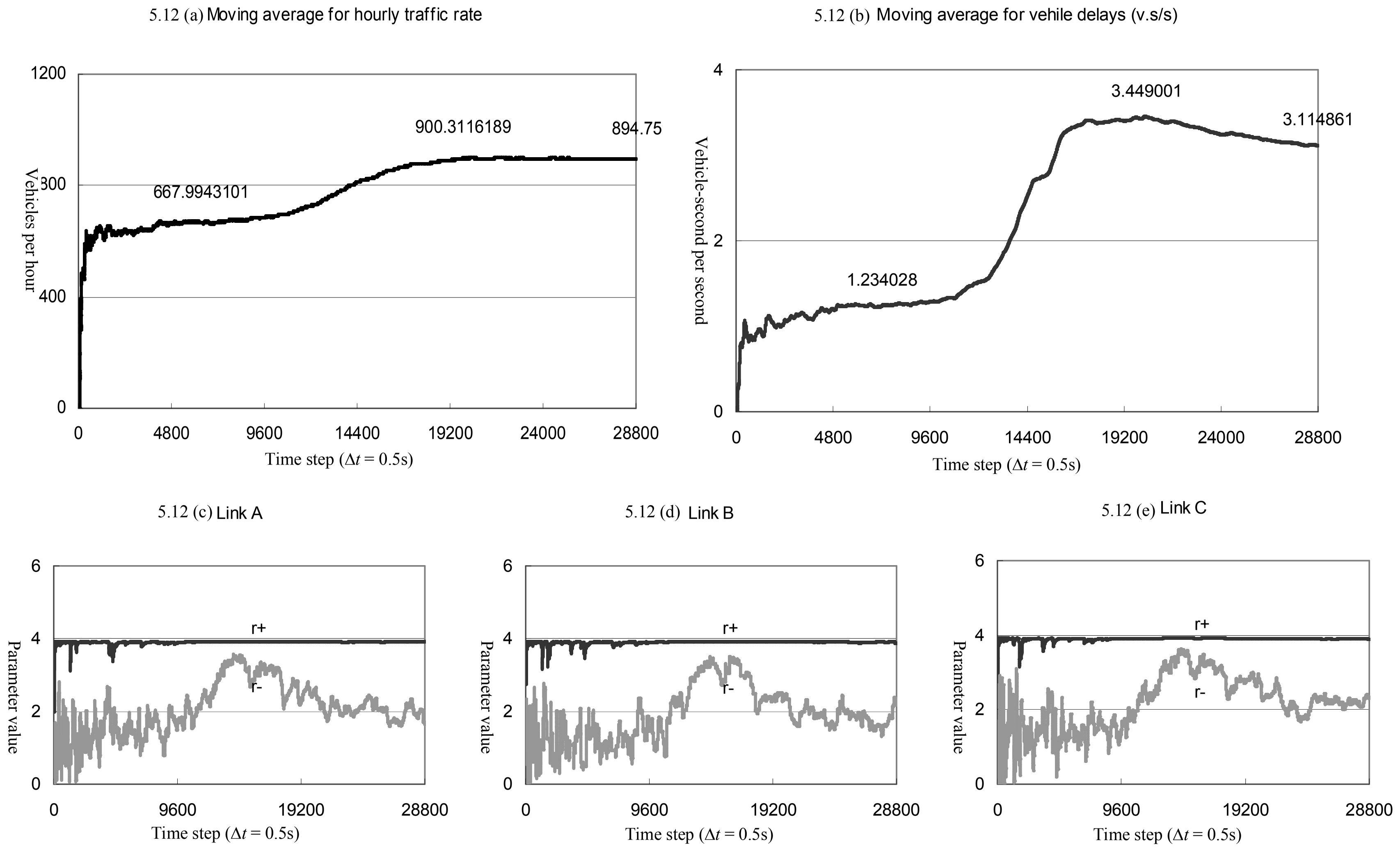
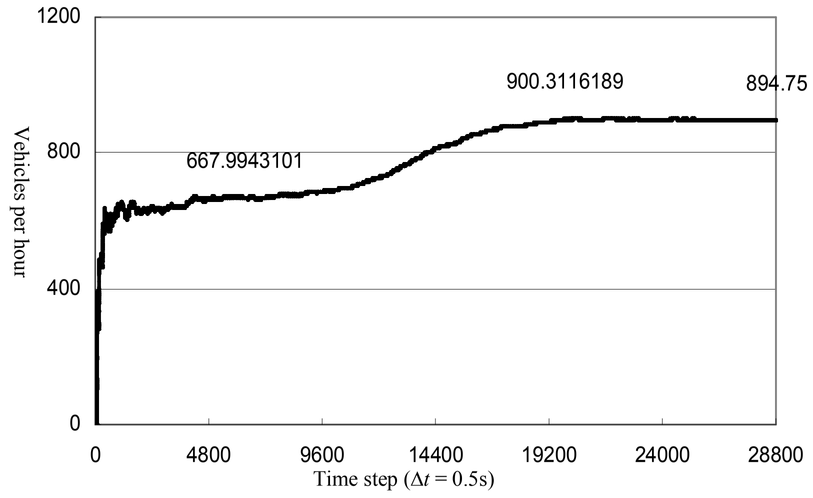
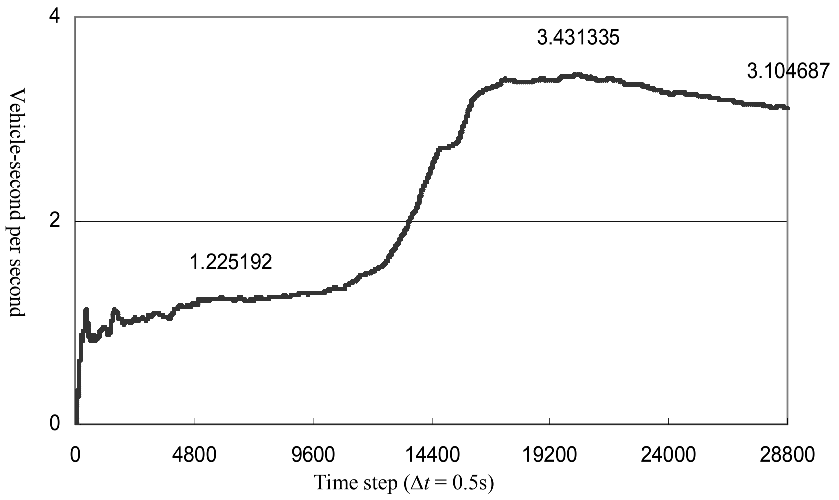


Fig. 5-12 Controller performance and approximation function parameters, ADP_PL, with non-stationary traffic

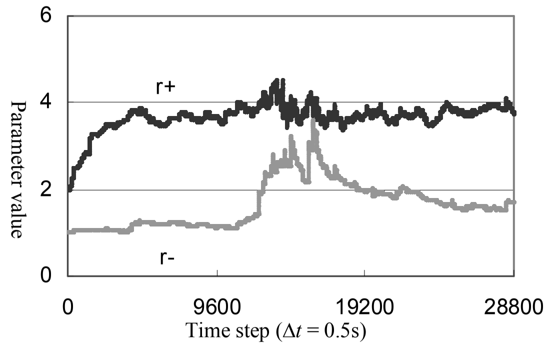
5.13 (a) Moving average for hourly traffic rate



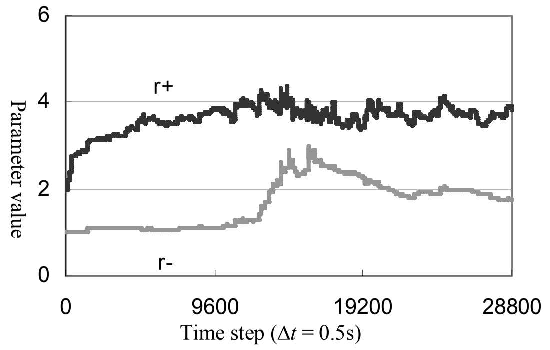
5.13 (b) Moving average for vehicle delays (v.s/s)



5.13 (c) Link A



5.13 (d) Link B



5.13 (e) Link C

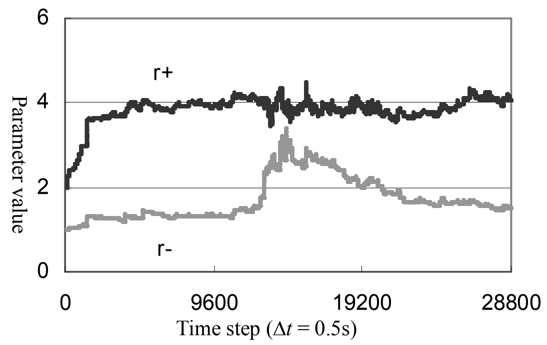


Fig. 5-13 Controller performance and approximation function parameters, ADP_TD, with non-stationary traffic

5.2.3.4 Influence of learning

We can investigate the effect of learning on performance by suspending the learning process (either temporal-difference learning or perturbation learning), thus only use the initial values of function parameters. The results thus obtained are then compared with the results from learning.

At 5.0s resolution, without learning, the averaged performance of ADP_PL over 20 runs is 10.78 v·s/s. Comparing to the result with learning (Table 5-7), which is 8.64 v·s/s, using the learning process produces about 8% reduction in vehicle delays.

At 0.5s resolution, with stationary traffic, the averaged performance over 10 runs is 4.64 v·s/s without learning. Using paired *t*-test, we found no significant difference between resutls obtained with learnings (Table 5-10) and without learning. With non-stationary traffic, the result is 3.07 v·s/s without learning, comparing to 3.10 v·s/s with ADP_TD and 3.11 v·s/s with ADP_PL. This suggests that at finer resolution, with a 20s planning horizon and a substantial discount rate, the influence of learning is insignificant to the ADP controller performs. Given the horizon of 20s and the discount rate of 24% per second, using Eq. (5-6), we have

$$\alpha^{20} = e^{-0.24\times 20} = 0.008 .$$

which means that only 0.8% of the output value of the approximation function is taken account. This is because in the value iteration of the approximate dynamic programming, we use Eq. (4-31) to compute:

$$\mathbf{u}_t^* = \arg\min E\left[\sum_{k=t}^{t+M-1} \alpha^t g\left(i_k, w_k, u_k\right) + \alpha^M \tilde{J}_{t-1}\left(i_{t+M-1}\right)\right],\ i \in X .$$

Since the learning techniques are used to update the approximation, their influences are substantially discounted too.

5.2.4 Summary

In this section, we presented a series of experiments that use ADP controllers for controlling isolated intersections. In the case of two-stage intersection, we found that ADP_PL controller using linear approximation function performs as well as the best of existing control strategies, except for BDP. In particular, the results suggested that using a linear function is equally effective as using some kinds of non-linear function to approximate the value function. In the case of three-stage intersection, we compared the coarse resolution at 5s per temporal increment with the fine resolution at 0.5s. At the coarser resolution, the ADP_PL controller produced acyclic signal sequences to accommodate variation in the random arrivals and this reduced vehicle delays by 48% on average from the best fixed-time signal plans. The parameters of the approximation function were updated progressively through perturbation learning. The learning process explained about 8% of the benefits achieved in the case of constant inflows. At the finer resolution, with stationary traffic, the ADP controllers reduced 42% delay from the coarser resolution. Despite the different method for machine learning, ADP_PL and ADP_TD were similar in performance.

With non-stationary traffic, the ADP controllers demonstrated their ability to adapt to changes in prevailing traffic. Regardless of the learning techniques, parameters r^- are more responsive to the changes in traffic, whilst r^+ is relatively stable. However, the learning process contributed little to gains in performance in the case of fine resolution.

5.3 Traffic Network

In this part of the research, we investigate controlling a small-scale distributed network. Key to our interest is whether the ADP controllers could implicitly achieve signal coordination between upstream and downstream, while ensuring good performance at local intersection.

5.3.1 Geometric layout of the network system

The sample traffic network is adopted from the study of Wong *et al.* (2007). The network system is formulated using the cell transmission model (Daganzo, 1994) with a temporal increment of 2s. The outline of the network is shown in Fig. 5-14. The network system contains 8 road links, in which slinks L3 and L8 are short links. Traffic signals are installed at the end of links L1, L2, L3, L6, L7, and L8 to control the conflicting movements involved; the two exit links L4 and L5 are excluded. The arrows represent the lane markings that show the directions that are permitted from different road links. Links L1, L2, L6, and L7 are input (source) links on which traffic demands are generated and enter into the signalized CTM system.

The design of this network presents a challenging problem. The short links L3 and L8 are the bottlenecks of the system, and each of them belongs to upstream and downstream intersections. Regarding this, coordination among signals is critical to reduce vehicle delays. Inappropriate signal timings may result in queue spilling-back in L8 and L3, and blocking the outflow from L6 and L1.

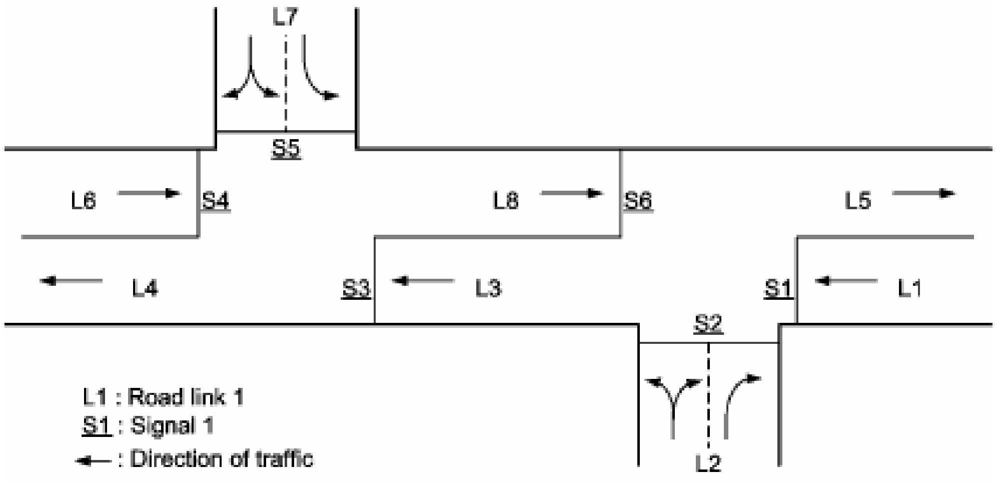


Fig. 5-14 A small-scale traffic network of two intersections

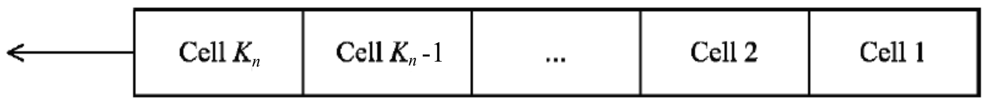


Fig. 5-15 Cell representation of road link *n* in the cell transmission model

Table 5-17 Modelling details of the traffic links in the example network

Intersection	Link	Total no. of Cells	Signal	Signal stage	Signal in Cell no.	Remarks
A	L1	5	S1	A1	5	Input link
	L2	5	S2	A2	5	Input link
	L5	3	-	-	-	Exit link
	L8	1	S6	A1	1	Short link
B	L3	1	S3	B1	1	Short link
	L4	3	-	-	-	Exit link
	L6	5	S4	B1	5	Input link
	L7	5	S5	B2	5	Input link

Table 5-18 Traffic inputs to the network system

Link	L1	L2		L6	L7	
Flow rate (vehicles per hour)	350	382		440	382	
Downstream	L3	L3	L5	L8	L4	L8
Turning ratio	100%	25%	75%	100%	25%	75%

Using CTM, each road link is segmented and represented by a series of homogenous cells, as shown in Fig.5-15. Vehicles that are scheduled to enter a road link are stored in the first cell (Cell 1) of that link. During each temporal interval, those vehicles that are already in the upstream cell can move to the next cell downstream. The amount of traffic that can proceed forward depends on the downstream spatial availability and the link saturation flow. Instead of moving forward to next downstream cells, vehicles must hold up and stay in the current cell if there is insufficient space available downstream. Upstream traffic will even be blocked and held up simultaneously. A physical vehicle queue will then develop to realise the congestion effects. Mathematical representation of the CTM model is given by Eq. (4-35) and (4-36) in Section 4.8.3.2.

The modeling details of the road links are given in Table 5-17. There are five cells used to model each input links (L1, L2, L6, and L7). Three cells are given for each exit links (L4 and L5). Road links L3 and L8 are defined as short links that contain only a single cell. Each cell has a holding capacity of 4 vehicles, except for the last cell in exit links, which has no limit to its holding capacity.

Links L1, L2, L5, and L8 constitute intersection A, while links L3, L4, L6, and L7 constitutes intersection B. Consequently, signals S1, S2, and S6 belong to intersection A, and signals S3, S4, and S5 belong to intersection B. There are two signal stages at intersection A, with S1 and S6 belonging to stage A1, and S2 to A2; and two signal stages at intersection B, with S3 and S4 belonging to stage B1, and S5 to B2. Each traffic intersection is governed by a local ADP controller. The operation of the local controllers follows the general assumptions and principles stated in Chapter 4. The mandatory inter-green and minimum green are set at 6.0s each (as we are using 2s temporal increment) in this case.

5.3.2 Input assumptions

In this experiment we only use stationary traffic arrival rate. The traffic flows in input links are generated using the inverse transformation method discussed in Section 4.2. The flow inputs and their downstream distributions are given in Table 5-18. Fixed proportions of 75% and 25% of input flow from L2 turn into downstream links L3 and L5 respectively. Similarly, 75% and 25% of input flow from L7 turn into downstream links L4 and L8 respectively. We also assume that links L2 and L7 contain a single shared lane for both left- and right- turn traffic and thus all turning flows will be blocked if either one of the associated downstream cells is fully occupied. With the configuration of the input and distribution pattern, the direction of west-to-east through links L6 to L8 and to L5 becomes the major channel of traffic, which accounts for 1013 vehicles per hour (100% of L6 plus 75% of L2 and L7).

5.3.3 Definition of queue in CTM

Although the CTM processes the distribution of traffic along links, the model itself does not estimate queue length in a link. Meanwhile, the feature-extraction function (Eq. 4-25 and 4-26), which is used as the basis function for the linear approximation function, takes queues of each link as input and differentiates them according to signal status. This requires a set of rules to define queue in the CTM.

Regarding the geometric layout of the sample network and the characteristics of the CTM model, we define queues according to the following rules:

1. For links L3 and L8, at the end of each increment,

   Let $l\ (n) = x\ (k)$, where $l\ (n)$ is the queue length and $x\ (k)$ the number of vehicles in cell k;

2. For each link of L1, L2, L6 and L7, at the end of each discrete time interval,

   2.a Set $k = K_n$, where K_n is the total number of cells in link n.

   2.b Let queue $l\ (n) = x\ (k)$.

   2.c For $k = K_n - 1, \ldots, 1$,

   If $x\ (k + 1) = C\ (k)$, where $C\ (k)$ is the holding capacity of cell k; or if $x\ (k) > 1$; or if $x\ (k) + x\ (k + 1) > C\ (k)$, let $l\ (n) = l\ (n) + x\ (k)$;

   Else, iteration terminates.

The resulting queue status is then processed by the feature-extraction function.

5.3.4 Control policy

Because the network system is modelled at 2s per temporal increment, the ADP controller consequently revises its plans at the same frequency. The local ADP controller considers the following possible decisions:

(d) The signals are not changed;

(e) The signals change immediately to the signal stage that gives least total delay;

(f) The signals change in $k\Delta t$ seconds, for $k = 1, 2, \ldots, T_h - 1$, 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 decision (a) and (c). Using the rolling horizon approach, only the first Δt, i.e. the first two seconds, is implemented. The optimal planning horizon for network control, however, is to be found in pilot tests. In this test, we only investigate ADP_TD method.

5.3.5 Performance

In pilot tests, we found that a horizon of 16s resolution is a favourable choice in general. We also found that $\theta = 20\%$ per second is the favourable value for discounting future delays.

We first obtain results from 10 independent simulation runs, each of 7200 temporal increments, equivalent to 4-hour simulated time. The results are shown in Table 5-19. The performance on average is 9.00 v·s/s with a standard diviation of 0.38 v.s/s. The competitiveness of this result is evidenced by the descriptive statistics of queues in Table 5-20. With the majority of traffic moves from west to east and a short link L8 of capacity 4 vehicles only, there were no substantial delays in links L6 (mean queue length 1.35) and L7 (mean queue length 2.32). This implies that any temporary long queue in L6 (max 11) and L7 (max 12) was quickly dissipated through coordinations between upstream and downstream signals, which belong to different cotnrollers.

The key to coordination is signal S6 that controls link L8. The ADP controller of intersection A proved effective in alternating S6 (S1) and S2 to accommodate upstream demand, while leaving no significant queues in local links L1 (mean 0.78) and L2 (mean 1.76). Similaly, the controller of intersection B is equally effective.

Table 5-19 Performance of using ADP_TD for distributed traffic network; each run contains 7200 temporal increments at $\Delta t = 2$s

Run	Performance (v.s/s)
1	8.99
2	8.67
3	9.49
4	9.41
5	8.76
6	8.81
7	8.71
8	8.90
9	8.58
10	9.66
Mean	9.00
SD	0.38

Table 5-20 Queuing statistics of a single simulation run of 4-hour simulated time, ADP_TD for network control

	Intersection A			Intersection B		
Link	L1	L2	L8	L3	L6	L7
Mean	0.78	1.76	1.93	0.84	1.35	2.32
SE	0.01	0.02	0.02	0.01	0.02	0.03
Median	0	1	2	0.75	1	2
Mode	0	1	3	0	0	0
Maximum	7	11	4	4	11	12

A sample visualisations of signal coordination among S4, S5 and S6 is shown in Fig. 5-16, and in Fig. 5-17 for S1, S2, and S3.

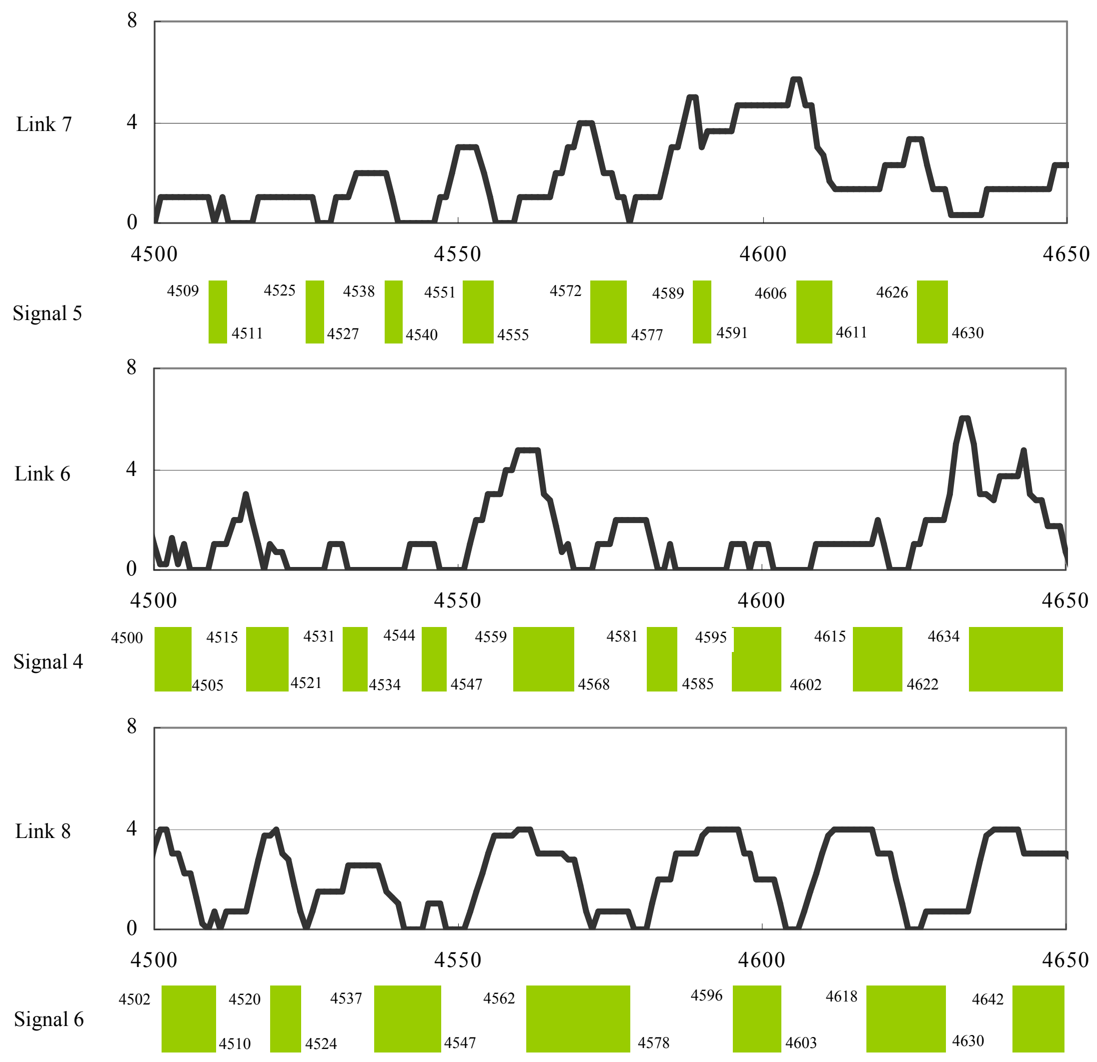


Fig. 5-16 Coordination between upstream signals S4, S5, and downstream signal S6; X-axis indicates time units, Y-axis indicates queue length in vehicle units

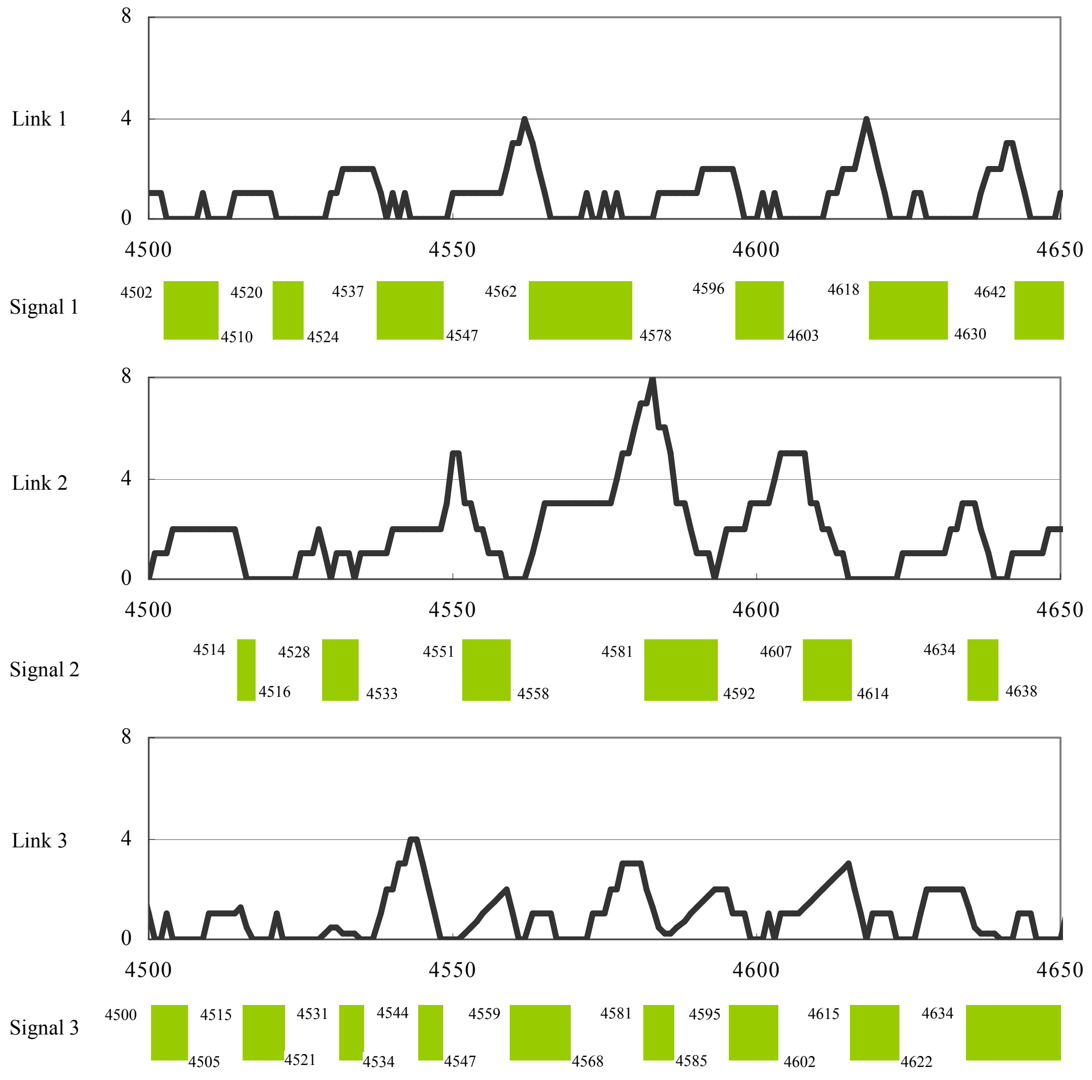


Fig. 5-17 Coordination between upstream signals S1,S2, and downstream signal S3; X-axis indicates time units, Y-axis indicates queue length in vehicle units

A typical case of coordination demonstrated in Fig. 5-16 is that controller at intersection A uses the holding capacity of L8 to accommodate the first few arrivals from upstream until fully occupied (e.g. t = 4551 to 4561), and then switch S6 to green to dissipate queues at saturation flow rate (1 v/2s) until flow rate drops (e.g. t = 4562 to 4578). This shows that the controller at intersection A maximises flow until upstream queues are cleared, at which point the incoming flow rate to L8 converges to arriving rate. In the mean time, using the holding capacity of L8 to accommodate the first few arriving vehicles gives the controlller opportunites to clear local queues in L2 (t = 4551 to 4558, and t = 4581 to 4592). The controller at intersection B coordinates flows from links L1 and L2 in a similar manner, despite the lower demand from east to west.

5.3.6 Parameters of approximation function

Each intersection is operated by an independent ADP controller. The learning rate for ADP_TD is set at $\eta = 0.001$ (for using Eq. 3-74) to update parameter estimation), which is the same as for the case presented in Secition 5.2.3.2. The evolution of approximation function parameters is plotted in Fig. 5-18.

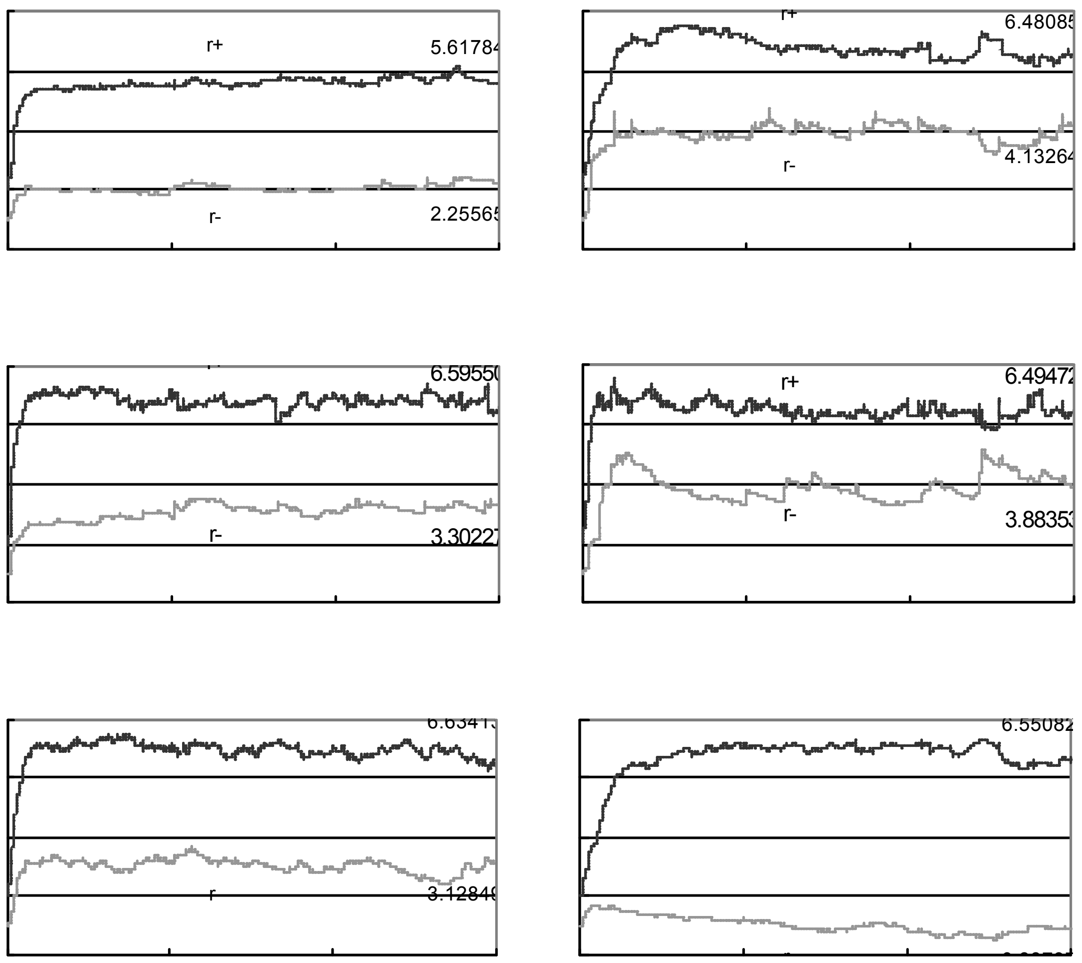


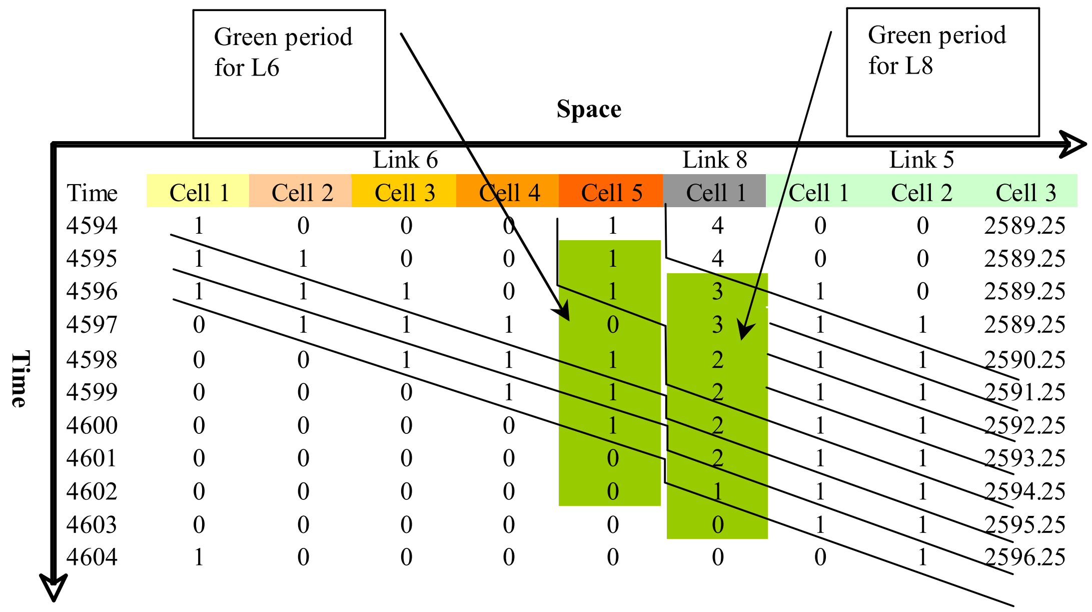
Fig. 5-18 Approximation function parameters in traffic network control using the CTM model; X-axis indicates time units, Y-axis indicates values of parameters

Under stationary traffic, parameters of Intersection A are relatively stable in the steady-state. The parameters of Intersection B, however, show greater variation than those of Intersection A. The mean values of r^+ of all links are similar to each other in the steady-state, except for that of link L1. Link L1 has the least traffic demand but longer green times (S1 and S6 share stage A1), and therefore vehicles in this link experience least delays in general, thus explaining the lower value of r^+. The values of parameter r^- seem less regular, and seem more related to the combined effects of local traffic demand and downstream condition. For example, L3 has moderate traffic demand but without downstream constraint, thus having lowest r^- of all. On the other hand, link L6 has the highest demand, and a short-link L8 downstream, thus having the greatest r^-, which means that vehicles queued in green in L6 spent more time on average to leave the intersection.

5.3.7 CTM in simulation

The CTM model allows physical queues to be modelled in simulation. The direct implication is that queues in L3 and L8 cannot exceed 4 vehicles, and queues may spill back to the upstream stoplines if they are poorly coordinated, or if the network is over-saturated. The experiment with ADP controllers shows that spilling-back of queue is broadly avoided in heavily traffic situation (but still less than full saturation). As shown in Fig. 5-16, if queue in L8 reaches 4 vehicles and S6 is not switched to green, the upstream controller normally switches green signal to other links that allows queue to dissipate; and the downstream controller normally switches S6 to green shortly after the maximum queue is reached in L8. This kind of coordination broadly avoids blocking vehicles leaving from upstream at green signal.

A sample of the propagations of vehicle in the CTM is shown in Fig. 5-19. Vehicles travel at free flow speed in low density area (L6 and L5), but are held in high density area (L8) until free flow speed propagation can be resumed. This pattern conforms to the definition of CTM, under which CTM is a discrete approximation of the LWR model.



	Link 6					Link 8	Link 5		
Time	Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 1	Cell 1	Cell 2	Cell 3
4594	1	0	0	0	1	4	0	0	2589.25
4595	1	1	0	0	1	4	0	0	2589.25
4596	1	1	1	0	1	3	1	0	2589.25
4597	0	1	1	1	0	3	1	1	2589.25
4598	0	0	1	1	1	2	1	1	2590.25
4599	0	0	0	1	1	2	1	1	2591.25
4600	0	0	0	0	1	2	1	1	2592.25
4601	0	0	0	0	0	2	1	1	2593.25
4602	0	0	0	0	0	1	1	1	2594.25
4603	0	0	0	0	0	0	1	1	2595.25
4604	1	0	0	0	0	0	0	1	2596.25

Fig. 5-19 A sample presentation of traffic propagations along the path from L6 to L5 via short link L8 in the CTM model, dark lines in the table represents vehicle trajectories.

5.3.8 TRANSYT plans

To compare with existing control strategies in network control, we use TRANSYT 12.0 to produce optimised fixed-time plans. The traffic network presented in Fig. 5-14 was reconstructed in TRANSYT with identical conditions, except for omitting links L4 and L5, which are exit-links. The data set for TRANSYT and the corresponding results are shown in Appendix 5.A.

By setting cycle time at 120s, and 2s per step, the optimised plans obtained from TRANSYT are shown in Table 5-21. The timings in the table correspond to the starting times of the stages in a cycle. Because of using double-cycle strategy, there are two green-time slots for each stage in a cycle. In response to the major flow from link L6 to L5 via L8, stage A1 and B1 share two common green periods in a cycle: 33 seconds in the first half of the cycle, and 24 seconds in the second half of the cycle, totaling at 57 seconds, 45 seconds of which are effective green. However, the common green shared between stage B2 and A1 is only 24 seconds in total, 12 seconds of which are effective green, despite that 75% of link L7's traffic turns to L5 via L8.

Table 5-21 The optimal fixed-time signal plans from TRANSTY 12.0 for the traffic network

Intersections	Number of Stages	Stage 1	Stage 2	Stage 3	Stage 4	Cycle time
A	4	119 (A1)	49 (A2)	73 (A1)	107 (A2)	120 seconds
B	4	2 (B1)	35 (B2)	67 (B1)	97 (B2)	120 seconds

Table 5-22 The performance of the TRANSYT plans in the CTM model

	Intersection A			Intersection B		
Link	L1	L2	L8	L3	L6	L7
Mean queue	0.86	70.14	2.26	0.81	2.05	34.36
SE	0.01	0.42	0.02	0.01	0.02	0.27
Median	0	67	3	0.25	1.25	27.67
Mode	0	66	4	0	0	21
Maximum	7	132	4	4	11	81.67

We implemented the fixed-time plans from TRANSTY to the CTM model, and the resulting performance measures are shown in Table 5-22. The main indicator of the performance is the mean queue length in each traffic link. The sum of mean queue length of all links is the average rate of vehicle delay (v·s/s). Although the TRANSYT plans maintain the queues in link L1 and L6 at low levels similar to the results from the ADP controller, the queues on L2 and L7 are simply enormous: vehicle delay in L2 is about 40 times greater than that from the ADP controllers, and delay in L7 is about 15 times greater than results obtained by the ADP controllers.

This comparison suggests that the signal plans calculated in TRANSYT using deterministic plantoon dispersion model, which assumes cyclic traffic profile, may not perform satisfactorily when link traffic dynamics are stochastic and heavily affected by the existence of capacity bottlenecks. In contrast, the ADP controller offers a competitive solution for operating signals in such environments.

5.3.9 Influence of learning

The influcence of learning can be assessed by suspending learning process in operations, and use only initial values of the parameters of the approximation functions. We obtained 10 performance results without learning and compared corresponding results obtained with learning in Table 5-23.

This paired comparison shows that the contribution of learning to performance is not significant, as in paired-*t* test we accept the null hypothesis of equal mean. The reasons for this are the same as we discussed for the influence of learning in Section 5.2.3.4.

Table 5-23 Performance of ADP_TD controllers in distributive network control, and comparisons with ADP_TD without learning

Run	ADP_TD (vs./s)	ADP_TD without learning (vs./s)
1	8.99	8.95
2	8.67	8.66
3	9.49	9.54
4	9.42	8.87
5	8.76	8.62
6	8.81	8.79
7	8.71	8.62
8	8.90	9.12
9	8.58	8.27
10	9.66	9.33
mean	9.00	8.88
SD	0.38	0.37

5.3.10 Summary

In this section we presented numerical experiments in applying the ADP controller to distributed network control. The sample network consists of two traffic intersections with two short links of limited holding capacity connecting the upstream and the downstream, and each intersection is controlled by an independent ADP controller. Traffic dynamics in the network are modelled by the cell transmission model (CTM). Despite the high degree of saturation and uneven flow pattern, delays expereienced by vehicles were kept low in the traffic system operated by the ADP controllers. There was no persistence of queue in any link during the simulation. By implementing the optimised fixed-time plans from TRANSYT, we found that the signal plans optimised under the assumption of cyclic traffic profile does not perform well in stochastic environment. This further strengthens the advantage of the ADP controller, which operates signals dynamically to accommodate stochastic arrivals. The influence of learning to performance was found insignificant.

Without explicit coordination rules, the good performance of the distributed ADP controllers suggests that link traffic dynamics convey sufficient information for distributed control. Monitoring and accurate modelling of link flows are important for effective control.

5.4 Discussion

This chapter presents a systematic assessment of the performance of ADP controllers in a range of numerical experiments. Simulations were designed to represent real-time traffic control. Performance results were obtained in a potofolio of test scenarios, including a two-stage isolated intersection, a three-stage isolated intersection at each of coarse resolution (5s) and fine resolution (0.5s), and a two-intersection traffic network. In the two-stage case, the ADP algorithm using a linear approximation function is as good in performance as a heuristic solution using non-linear approximation function (Robertson and Bretherton, 1974). In the three-stage case at the coarse resolution, the ADP controller achieved 48% reduction in delay from optimised fixed-time plans. The controller's performance was further improved by 42% to just 33% of the original rate of delay (from optimised fixed-time) at the fine resolution. In the case of the small-scale traffic network with ADP controllers operating distributedly, the controllers consistently outperformed optimised fixed-time plans produced by TRANSYT 12.0.

Most of the benefits from ADP controllers are attributed to the features of improved awareness of traffic state, real-time sequential decision-making, and evolutionary approximation to value function. On the other hand, fixed-time plans are calculated under the rigid principle of equal degrees of saturation for all signal stages, and the assumption of stationary traffic or cyclic platoon arriving profiles. This explains broadly why fixed-time plans worked poorly in simulations of random arrivals, whereas ADP controller performed well.

In all cases, we used linear approximation function, and the learning techniques used for updating function parameters are relatively simple and straightforward in computation. In most cases, function parameters stabilised with bounded fluctuation after the system entered steady-state. The time the controller takes to complete one iteration of the ADP algorithm is trivial, suggesting that the ADP controller could be implemented for real-time operation without significant upgrade of hardware in signal controller.

Serveral interesting findings emerged from the experiments. The first is that a planning horizon of 10 – 20s is sufficient for good performance. This reaffirms conclusions drawn in previous studies (Robertson and Bretherton, 1974; Gartner, 1982; Bell *et al.*, 1990). Traffic information for the horizon can be supplied adequately by existing detection technologies, providing that the detectors are installed sufficiently upstream of the stopline. In cases where detected information fall short of demand, traffic models can be used to supplement the reminder of the planning horizon. However, in this study, we did not model the situation of queue spilling back over upstream detectors, in which case direct detection of arriving traffic is no longer possible. Predicting end of queue as queue spills beyond detectors is a case of significant pratical interest and demands further investigation.

Second, we found that, as resolution improves and horizon increases, the ADP controllers favour larger discount rate. In the cases of 5s resolution where horizon is set at 10s, discounting future delays at 0.8% per second works best, whilst in the cases of fine resolution where horizons are set between 16.0 – 20.0 seconds, discount rates between 20 – 24% per second work best. This highlights the importance of accurate monitoring and modelling traffic dynamics at present and in the immediate future. It suggests that improving detection technologies and better estimation of queue status are more cost-effective than exploring advanced approximation methods for the value function, whose outcome becomes insignificant if being discounted substaintially over time.

Third, in the cases of fine resolution with substaintial discount, controller performance are indifferent to learning techniques. This can be explained by the followings arguments: given that learning processes exercise influence on control through the approximation function, and because control performance is not sensitive to approximation outcome in case of substaintial discount, performance is broadly unaffected by the learning process.

Fourth, in the case of network control, the ADP controllers proved better equipped for operating at stochastic environment than optmised fixed-time plans, which are calculated from deterministic models under the assumption of cyclic traffic profiles. The resutls also showed that coordinations among distributed controllers can be achieved inplicitly by using appropriate traffic models that convey traffic information in the network. A moderate length of horizon, such as 16s, and a rapid revision of signal plans ensure that coordination is established effectively.

Appendix 5.A TRANSYT 12.0 Outputs for the Traffic Network Control

```
                                              TRANSYT 12.0

                                             Chen's Network


 PARAMETERS CONTROLLING DIMENSIONS OF PROBLEM :
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 NUMBER OF NODES                              =   2
 NUMBER OF LINKS                              =   6
 NUMBER OF OPTIMISED NODES                    =   2
 MAXIMUM NUMBER OF GRAPHIC PLOTS              =   0
 NUMBER OF STEPS IN CYCLE                     =  60
 MAXIMUM NUMBER OF SHARED STOPLINES           =   0
 MAXIMUM NUMBER OF TIMING POINTS              =   4
 MAXIMUM LINKS AT ANY NODE                    =   3


CORE REQUESTED =   5167 WORDS
CORE AVAILABLE =  72000 WORDS

DATA INPUT :-
      ~~~~ ~~~~~
CARD   CARD
 NO.   TYPE
(  1)= TITLE:- Chen's Network
CARD   CARD   CYCLE  NO. OF   TIME EFFECTIVE-GREEN  EQUISAT 0=UNEQUAL FLOW   CRUISE-SPEEDS   OPTIMISE  EXTRA  HILL-   DELAY   STOP
 NO.   TYPE   TIME    STEPS  PERIOD DISPLACEMENTS  SETTINGS  CYCLE    SCALE    SCALE  CARD32   0=NONE   COPIES  CLIMB    VALUE    VALUE
                PER   1-1200  START    END    0=NO  1=EQUAL  10-200  50-200 0=TIMES  1=O/SET   FINAL  OUTPUT  P PER   P PER
           (SEC)  CYCLE   MINS.  (SEC)   (SEC)    1=YES  CYCLE      %       %   1=SPEEDS 2=FULL    OUTPUT  1=FULL  PCU-H    100
  2)=  1    120     60    240     1     1     1     0     0     0     0     2     0     0   1420    260
CARD   CARD                           LIST  OF  NODES  TO  BE  OPTIMISED
 NO.   TYPE
  3)=  2     1     2     0     0     0     0     0     0     0     0     0     0     0     0     0

                        NODE CARDS:  MINIMUM STAGE TIMES (WORKING)
CARD   CARD   NODE          S1     S2     S3     S4     S5     S6     S7     S8     S9    S10
 NO.   TYPE    NO.
  4)=  10      1             6      6      6      6
  5)=  10      2             6      6      6      6

                        NODE CARDS:  PRECEDING INTERSTAGE TIMES (WORKING)
CARD   CARD   NODE          S1     S2     S3     S4     S5     S6     S7     S8     S9    S10
 NO.   TYPE    NO.
  6)=  11      1             6      6      6      6
  7)=  11      2             6      6      6      6

                        NODE CARDS:  STAGE CHANGE TIMES (WORKING)
CARD   CARD   NODE  Sgl/Dbl  S1     S2     S3     S4     S5     S6     S7     S8     S9    S10
 NO.   TYPE    NO.  Cycled
  8)=  12      1      1      0     49     75    105
  9)=  12      2      1      0     35     65     95

                              LINK  CARDS:   FIXED  DATA
                         FIRST   GREEN              SECOND  GREEN
CARD   CARD   LINK    EXIT        START         END          START          END       LINK    STOP     SAT   DELAY   DISPSN
 NO.   TYPE    NO.    NODE    STAGE    LAG    STAGE    LAG    STAGE    LAG    STAGE    LAG  LENGTH  WT.X100  FLOW  WT.X100    X100
 10)=  31      1      1      1      0      2      0      3      0      4      0    120      0   1440      0      0
 11)=  31      2      1      2      0      3      0      4      0      1      0    120      0   1440      0      0
 12)=  31      3      2      1      0      2      0      3      0      4      0     24      0   1440      0      0
 13)=  31      6      2      1      0      2      0      3      0      4      0    120      0   1440      0      0
 14)=  31      7      2      2      0      3      0      4      0      1      0    120      0   1440      0      0
 15)=  31      8      1      1      0      2      0      3      0      4      0     24      0   1440     10      0

                              LINK  CARDS:    FLOW DATA
                         ENTRY 1 ............   ENTRY 2 ............   ENTRY 3 ............   ENTRY 4 ............
CARD   CARD   LINK   TOTAL  UNIFORM   LINK         CRUISE   LINK         CRUISE   LINK         CRUISE   LINK         CRUISE
 NO.   TYPE    NO.    FLOW     FLOW     NO.    FLOW    TIME     NO.    FLOW    TIME     NO.    FLOW    TIME     NO.    FLOW    TIME
 16)=  32      1    350      0      0      0     10      0      0      0      0      0      0      0      0      0
 17)=  32      2    382      0      0      0     10      0      0      0      0      0      0      0      0      0
 18)=  32      3    450      0      1    350      2      2     95      2      0      0      0      0      0      0
 19)=  32      6    440      0      0      0     10      0      0      0      0      0      0      0      0      0
 20)=  32      7    382      0      0      0     10      0      0      0      0      0      0      0      0      0
 21)=  32      8    730      0      6    440      2      7    287      2      0      0      0      0      0      0

                            LINK DATA:  QUEUE CONSTRAINTS
CARD   CARD   LINK    LIMIT  QUEUE   LINK    LIMIT  QUEUE   LINK    LIMIT  QUEUE   LINK    LIMIT  QUEUE   LINK    LIMIT  QUEUE
 NO.   TYPE    NO.    QUEUE  WEIGHT    NO.    QUEUE  WEIGHT    NO.    QUEUE  WEIGHT    NO.    QUEUE  WEIGHT    NO.    QUEUE  WEIGHT
 22)=  38      3      4     20      8      4  10000      0      0      0      0      0      0      0      0      0

*****END OF SUBROUTINE TINPUT*****
  120 SECOND CYCLE  60 STEPS

FINAL SETTINGS OBTAINED WITH INCREMENTS :-   18  48  -1  18  48   1  -1   1
 - (SECONDS)

  NODE   NUMBER   STAGE   STAGE   STAGE   STAGE   STAGE  STAGE   STAGE   STAGE   STAGE   STAGE
   NO   OF STAGES   1       2       3       4       5       6       7       8       9      10

   1      4     119     49     73    107
   2      4       2     35     67     97
```

```
LINK    FLOW   SAT   DEGREE  MEAN TIMES -------DELAY--------  ----STOPS----  ----QUEUE----  PERFORMANCE   EXIT   GREEN TIMES
NUMBER  INTO  FLOW    OF     PER PCU   UNIFORM RANDOM+ COST   MEAN   COST   MEAN          INDEX.      NODE  START  START
        LINK        SAT   CRUISE           OVERSAT  OF    STOPS    OF     MAX.  AVERAGE  WEIGHTED SUM          END    END
                     DELAY  (U+R+O=MEAN Q) DELAY   /PCU   STOPS          EXCESS  OF ( ) VALUES        1ST    2ND
     (PCU/H) (PCU/H)  (%)  (SEC) (SEC)   (PCU-H/H)   ($/H)   (%)   ($/H)   (PCU)  (PCU)    ($/H)            (SECONDS)

   1    350   1440    35  10.5   7.2    0.4 +  0.3  (  9.3)    36  (  3.3)     3                  12.5         1   119  49  73   107
   2    382   1440    88  10.5  56.4    2.2 +  3.7  ( 84.2)   138  ( 13.6)   11                   97.8         1    49  73  107 119
   3    450   1440    60   2.5  14.3    1.0 +  0.7  ( 24.5)    63  (  7.3)     7  ( 0.2)*         31.9         2     2  35  67   97
   6    440   1440    58  10.5  16.1    1.2 +  0.7  ( 27.0)    69  (  7.8)     6                  34.9         2     2  35  67   97
   7    382   1440    56  10.5  17.8    1.2 +  0.6  ( 26.0)    73  (  7.2)     5                  33.2         2    35  67  97    2
   8    730   1440    72   2.5  11.0    0.8 +  1.3  ( 30.2)*   49  (  9.2)    10  ( 0.7)*         81.7         1   119  49  73   107

  TOTAL        TOTAL      MEAN        TOTAL    TOTAL    TOTAL       TOTAL        PENALTY       TOTAL
 DISTANCE       TIME    JOURNEY     UNIFORM   RANDOM+     COST        COST          FOR       PERFORMANCE
TRAVELLED      SPENT     SPEED        DELAY   OVERSAT     OF           OF          EXCESS       INDEX
                                      DELAY    DELAY     STOPS       QUEUES
(PCU-KM/H)   (PCU-H/H)   (KM/H)    (PCU-H/H)(PCU-H/H)   ($/H)        ($/H)        ($/H)        ($/H)

   214.8        19.1      11.2         6.8      7.3    ( 174.0) + (   48.5)  +  (  69.5)   =    292.0     TOTALS
------------------------------------------------------------------------------------------------------------------------
                                                                 ROUTE


***************************************************************************************************************************
                         CRUISE              DELAY              STOPS             TOTALS
                    LITRES PER HOUR    LITRES PER HOUR    LITRES PER HOUR    LITRES PER HOUR

FUEL CONSUMPTION PREDICTIONS        11.5        +       16.3        +       22.1        =       49.9

NO. OF ENTRIES TO SUBPT  =    5
NO. OF LINKS RECALCULATED=   24

PROGRAM TRANSYT FINISHED
============================================ end of file ============================================
```

CHAPTER 6 CONCLUSIONS

This chapter includes four sections. A concise summary of this doctoral study is provided in Section 6.1. Contributions of this study to methodological development and practice in traffic engineering are highlighted in Section 6.2. Main criticisms arising from this study are discussed in Section 6.3. Suggestions for further research on related topics are provided in Section 6.4.

6.1 Summary of Study

This study investigates approximate dynamic programming (ADP) for traffic signal control, aiming to develop a self-sufficient adaptive controller for real-time operation. The key feature of the proposed ADP approach is to replace the exact value function of dynamic programming (DP) with a linear approximation function. The approximation function is progressively updated by using machine-learning techniques. We have shown in numerical examples that at the resolution of 5s per temporal increment, the ADP controller reduced vehicle delays by 43% from the optimised fixed-time plans produced by TRANSYT 12.0. At the resolution of 0.5s per temporal increment, the ADP controller achieved a 67% reduction in vehicle delays from TRANSYT, and 42% better than the results at 5s resolution. We used the DP to produce the absolute higher bound in performance measure, and the ADP controller only caused 8% more delays. Additionally, the time the ADP controller takes to compute an hour's simulation is only about 0.08% of the time the DP approach takes to compute 6 minutes' simulation. The ADP controllers only assume 10s information of future arriving traffic, whereas the DP controller is provided with the complete information.

These results suggest that the ADP controller can achieve a large proportion of the benefits of DP, while being efficient in computation and practical for implementation.

In the case of the small-scale traffic network, we found that the distributed ADP controllers maintained delays low in all links. Active coordination between the upstream and downstream controllers was observed. Apart from the benefits achieved the ADP controllers, the cell transmission model (CTM) proved better for modelling traffic dynamics in network. Using the optimised TRANSYT plans to control the CTM based network caused 10 times more delays than the distributed ADP controllers.

The ADP algorithm requires machine-learning techniques to update approximation function progressively. 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, and propagates the difference back to the parameters of the approximation function. The approximation is consequently improved. Perturbation learning directly calculates the gradients of the approximate function by perturbing the system state. Despite of the different learning methods, the learning effects were broadly similar and no statistical difference in performance was found in numerical experiments. The ADP controller with either of the two learning techniques produced the best performance by discounting future delay at about 24% per second at the 0.5s resolution. The substantial discount suggests output of the approximation function is insignificant to influence decision-making, thus the influence of learning is limited. Additionally, given the good performance at this discount rate, it suggests that a simple linear approximation is sufficient for real-time control. Exploring more complex approximations may not prove cost-effective.

6.2 Contributions

This study presents the first systematic investigation in applying ADP to traffic signal control. The ADP concept offers a general solution to sequential-decision making in complex processes, where the sizes of the state space, information space and action space restrict direct application of classic dynamic programming. This study has developed ADP from its general concept to address adaptive traffic signal control as a specific case. For such a case, we systematically established the definitions for system state, dynamics of state transition and the structure of cost functions. The definitions are applicable for typical road intersections in both urban and rural area.

Several important structural properties of the value function have been identified for the development of ADP solutions. The non-decreasing monotonicity of the value function in queue length and signal state, as stated in Theorem 4.1, 4.2 and 4.3, have been proved and presented for transport studies for the first time. Based on the identified structural properties, we use linear approximation function and the feature-extraction function to differentiate signal status. The structural properties of the value function can be the basis for further investigation in function approximation, policy improvement and development in machine learning techniques for adaptive traffic signal control.

Several machine-learning techniques have been studied and employed to update parameters of the approximation function. In real-time control, function parameters are usually not known a priori. This makes the investigation in appropriate machine learning techniques, especially unsupervised learning, important for developing adaptive controllers. As the traffic environment evolves, the adaptive controller adjusts parameters accordingly. This study considered two learning techniques: the temporal-difference (TD) learning and perturbation learning. Theorem 3.2 (Section 3.4.4) guarantees that, using TD learning, estimates of the parameters of a linear approximation function converge with probability of 1. Alternatively, the linear separable structure of the function allows numerical calculation of gradients by perturbing system state. Both techniques update function parameters incrementally and are appropriate for real-time implementation. The formulae of the ADP algorithm allow a variety of learning techniques to be considered for further studies.

This study has direct and important implications for traffic engineering. The ADP algorithm has been purposely developed to address practical issues in traffic engineering. Usual control constraints, such as inter-green and minimum green, are built into the state transition functions. The ADP controllers are in compliance with a range of discrete systems of resolutions from 0.5s to 5s per temporal increment. There is no difficulty in principle to refine the resolution further to, say 0.1s, because the issues that arise when the time increment is shorter than control steps such as minimum green and clearance times have already been addressed in implementation of the present fine resolution. By using a feature-extraction interface, the controller is able to work with different traffic models, such as the vertical queuing model and cell transmission model. Traffic arrivals of the next 10s are assumed available from upstream detectors, which is a fair representation of the state-of-the-art in traffic control. Even if this reasonable assumption is not satisfied, the controller can easily use Monte Carlo simulation to support decision-making. The ability of the ADP control algorithm to address practical issues underscores its readiness for implementation.

In this study, the control objective is to minimise vehicle delays. The framework of ADP algorithm proposed here is capable of including vehicle stops and exhaust emission into objective function. Weighting factor can be applied to vehicle stops and emission rates so that traffic engineers can assign priority to specific control objectives.

We have proposed a fully distributed control system for traffic network operation. We showed in numerical examples that the ADP controller managed stochastic traffic arrivals substantially better than the fixed-time plans produced by TRANSYT 12.0. This highlights the potential benefits of using distributed adaptive controller in place of existing systems.

Overall, this study has developed an appropriate solution for practical application in traffic engineering based on approximate dynamic programming and machine-learning techniques. The presented ADP method, when implemented with fine temporal resolution, can achieve the majority of the benefits in control performance that are possible as assessed by full solution of a dynamic optimisation using BDP methods. The potential benefits of this approach are evidenced by its competitiveness in performance, efficiency in computation and readiness for practical implementation.

6.3 Critique

In machine-learning process, the stepsize (or the learning rate) scales new estimates of parameters to correct existing estimates. The convergence of using temporal-difference (TD) learning to adjust parameters of a linear approximation function, by Theorem 3.2 (Section 3.4.4), requires a diminishing stepsize satisfying Assumption 3-6 (Section 3.4.4). Considering the evolving traffic environment, and in concern of “over-shooting” in learning, we used a constant and cautious stepsize for the TD learning in this study. For perturbation learning, we used deterministic and diminishing stepsizes. In Chapter 5, we showed that in a static traffic environment, the parameters trained by TD learning with constant stepsize exhibited bounded variations in steady-state. In the same context, despite the diminishing stepsize, parameters for green signal status exhibited greater variation when using perturbation learning other than TD learning. In an evolving traffic environment, the parameter values exhibited similarity in evolution, despite the difference in learning techniques and stepsize rules. Adaptive stepsize is not considered in this study. Future studies may consider convergence of approximation with adaptive stepsizes.

An interesting discovery from the numerical experiments is that the ADP controllers favoured a substantial discount rate (20% ~ 24% per second) for future delays at fine resolutions. This has two implications: first, the impact of the cost-to-completion represented by the approximation function is limited on decision-making; second, given the limited impact of terminal cost, the difference in learning techniques does not have a substantial influence on performance. The second statement is evidenced by a comparison between TD learning and perturbation learning. This suggesets that using more sophiscated approximation structures and learning techniques may not prove cost-effective.

In Chapter 4, we proved a few key structural properties of the value function, and pointed out that the linear function defined by (4-25) to (4-28) provides a good approximation to the exact value function. However, because a traffic link can only have one of the two signal statuses, the parameters r^+ (assigned to red status) and r^- (assigned to green status) cannot be updated simultaneously. Additionally, the time spent in red and green status will not generally be equal. The frequency of update will therefore generally differ, and so is the impact on the evolution of parameters. This has been evidenced by the different degrees of variation in r^+ and r^- in numerical examples.

The traffic models adopted for this study are macroscopic and do not model the behaviour of individual vehicles. The traffic is assumed to be homogenous, and pedestrians are not considered. The objective function takes account of vehicle delays only, though it would not be difficult to incorporate pedestrian delays and vehicle stops as additional factors. The study on network control is relatively simple, with only a pair of intersections connected by two short-links, although this does highlight the issue of coordination because of the risk of queue blocking-back. Hierarchical structure of network control is not considered.

In this study we assumed perfect detector information, i.e. no error in detecting vehicles. In reality, because of specific characteristics of vehicles, there might be false alarms from detector or miss-detection, which may result in inaccurate representation of traffic state. Sensitivity of ADP controller to imperfect detector information is not investigated in this work. Knowledge from previous studies (Webster, 1957) suggests that signal control performance is more sensitive to underestimation of queue length than overestimation of queue length.

6.4 Future Works

The doctoral study identifies several areas that seem promising for further research. We discuss some of them here.

Approximate dynamic programming is a general concept that can incorporate a variety of approximation and machine-learning techniques. The continuous function approximation using reinforcement learning is one of many possibilities. We identify three areas for future investigation in approximation methods.

The first is the issue of dimensionality. By using linear approximation functions, we reduce the dimension of the control problem to a few functional parameters. Another common way of reducing dimensionality is state aggregation, for example representing queuing state as “low”, “medium” and “high”, traffic flows as “light” and “heavy”. By classifying the aggregated states, we may use specific methods, such as *Q-learning* (Watkins, 1989) to update the state-to-action mapping.

The second area is the architectures of approximation function. In Chapter 3 we introduced the concept of neural-dynamic programming, in which the neural network provides the learning capacity. By Theorem 3.1 (Section 3.4.1), the non-linear neural network, or the multi-layer perceptron (MLP), is a universal approximator to any function. This suggests that we may use MLP to approximate the value function in traffic signal control. The MLP is a feed-forward network trained by backward propagation. A step further from this is *recurrent neural network* (RNN), where connections between neurons form directed cycle. RNN can be trained by *back-propagation through time* (Werbos, 1990). The extension to MLP and RNN expand the choices of approximation structures and learning techniques. However, there is no proof so far that non-linear approximations produce convergence in functional parameters.

The third area is the stepsize. In this study we used constant stepsize for TD learning and deterministic stepsizes for perturbation learning. Ideally, we would expect the stepsize to be adaptive to the noise in the environment, i.e. the greater the noise the smaller the stepsize. A notable study in adaptive stepsize in approximate dynamic programming is by Powell (Section 6.3, pp.190, 2007).

For the exercise in traffic engineering, it will be worthwhile to compare the ADP controller with other contemporary adaptive signal controllers using independent micro-simulation software, such as Paramics and VISSIM (PTV Planning Transport Verkehr AG., 2004). The MOVA system will be a good candidate for benchmarking. Further from this point, a field test can be organised. The state transition function and the control algorithm may need further modification to accommodate the local environment, and further constraints may be introduced, such as constrains on the maximum green/red time and the stage/phase sequence. The implementation of the ADP controller may also require pedestrian phases, which have not been included in this study.

The preliminary study on distributed traffic network control reveals the potential of the ADP controller for larger scale operation. Further studies in this may extend the scale of the network to the size of a central urban area, and test the behaviour of the controllers with specific traffic scenarios, including not only peak periods but also priority schemes.

6.5 Final Remark

Adaptive traffic signal control is a challenging area that synthesises of control theories, queuing theories, traffic theories, and machine-learning methods. Notwithstanding its significance in theoretical development, traffic signal control is after all a practical issue that relates to daily life in urban areas as well as in remote locations. Although previous studies have shown that dynamic programming is a method to calculate the optimal solution, it has several practical limitations. This gives the issue of the extent to which a more practical but sub-optimal methodology such as ADP can approach to the optimum. Bearing in mind the practical interest, the sub-optimal solutions should be easy to implement, efficient in computation, and cost-effective to operate. In this study we have shown that approximate dynamic programming provides a realistic approach to address the issues of interest.

REFERENCES

Ahn, W.Y., 2004. Dynamic optimisation for isolated road junctions, Ph.D thesis, University of London.

Allsop, R.E. (1971a) Delay-minimising settings for fixed-time traffic signals at a single road junction, *Journal of the Institute of Mathematics and its Applications*, **8** (2), 164-185.

Allsop, R.E. (1971b) SIGSET: a computer program for calculating traffic signal settings, *Traffic Engineering and Control*, **13**, 58-60.

Barto, A.G., Bradtke, S.J., Singh, S.P. 1995. Learning to act using real-time dynamic programming, *Artificial Intelligence*, **72**, 81-138.

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.

Becker, S., 1991, Unsupervised learning procedures for neural networks, *International Journal of Neural Systems*, vol.2, 17-33.

Bell, M.G.H, Cowell, M.P.H, Heydecker, B.G., 1990. Traffic-responsive signal control at isolated junctions. In: Yagar, S., Rowe, E. (Eds.), Traffic Control Methods. New York: Engineering Foundation, 273-294.

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

Benveniste, A., Metivier, M., Priouret, P. 1990. Adaptive Algorithms and Stochastic Approximations, Springer-Verlag, Berlin.

Bertsekas, D.P., Nedich, A., Sorkar, V.S. 2004. Improved temporal difference methods with linear function approximation. In: Si, J., Barto, A.G., Powell, W.B., Wunsch, D. (Eds.), Handbook of Learning and Approximate Dynamic Programming, Wiley-IEEE Press, ISBN: 978-0-471-66054-5.

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

Boyan, J.A. 2002. Technical update: Least-squares temporal difference learning, *Machine Learning*, **49**, 1-15.

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

Cybenko, G. Approximations by superpositions of sigmoidal functions. Mathematics of Control, Signals, and Systems, 2:303–314, 1989.

Daganzo, C.F., 1994. The cell transmission model: a dynamic representation of highway traffic consistent with the hydrodynamic theory, *Transportation Research,* **28B**(4), 269-287.

Dayan, P.D. 1992. The convergence of TD(λ) for general λ, *Machine Learning*, **8**, 341-362.

Department of Transport. 1981. General principles of control by traffic signals, TA16/81, U.K.

Department of Transport. 1984. MCE 0141: Microprocessor based traffic signal controller for isolated linked and urban traffic control installations, London.
Department of Transport. 2009. Transport Trends 2008 Edition. London, United Kingdom.

Devroye, L. 1986. *Non-Uniform Random Variate Generation*. New York: Springer-Verlag.

Ferrari, S., Stengel, R. 2004. Model-based adaptive critic designs, in *Handbook of Learning and Approximate Dynamic Programming,* Jennie Si, J., Barto, A.G., Powell, W.B., Wunsch, D. (Eds.), 65 – 95, Published by Wiley-IEEE, 2004, ISBN 047166054X.

Ferrari, S., Stengel, R. 2004. Online adaptive critic flight control, *Journal of guidance, control and dynamics,* **27**(5), 777-786.

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.

Gartner, N.H., Messer, C., Rathi, A.K., 1997. A revised monograph on traffic flow theory, Federal Highway Administration Research, U.S. Department of Transportation. http://www.tfhrc.gov/its/tft/tft.htm

George, A.P., Powell, W.B. 2006 Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming, *Machine Learning*, **65**(1), 167-198.

Goodwin, P. 2004. The economic cost of road traffic congestion, discussion paper, the Rail Freight Group, London, UK.

Gordon, G.J. 1995. Stable function approximation in dynamic programming, *Technical Report: CMU-CS-95-103,* Carnegie Mellon University.

Haykin, S. 1999. Neural Networks: A Comprehensive Foundation, 2nd Edn, Prentice-Hall, Upper Saddle RIVER, NJ.

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. 2004. Objectives, stimulus and feedback in signal control of road traffic, *Intelligent Transportation Systems,* **8**, 63–76.

Heydecker B.G., Boardman, R.M. 1999. Optimisation of timings for traffic signals by dynamic programming, in *Proceedings of the 31st UTSG Annual Conference*, U.K.

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.

Heydecker, B.G., Dudgeon, I.W. 1987. Calculation of signal settings to minimise delay at a junction, in *Proceedings of 10th International Symposium on Transportation and Traffic Theory*, MIT, July, 159-178.

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.

Improta, G. and Cantarella, G.E. 1984. Control system design for an individual signalized junction. *Transportation Research*, **18B** (2), 147-167.

INFRAS/IWW. 2004. External Cost of Transport: Update Study, Final Report. Zürich/Karsruhe.

Kimber, R.M., Hollis, E.M. 1979. Traffic queues and delays at road junctions. Transport and Road Research Laboratory *Report* **LB909**. Crowthorne: TRL.

Kulkarni, N., Phan, M.Q. 2003. Optimal feedback control of a magneto-hydrodynamic generator for a hypersonic engine, AIAA-2003-5497, in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Austin, TX.

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.

Lighthill, M.J., and Whitham, J.B., 1955. On kinematic waves. I. Flow movement in long rivers. II. A theory of traffic flow on long crowded road. *Proceedings of the Royal Society A229*, 281-345.

Little, J.D.C. 1964. The synchronisation of traffic signals by mixed-integer linear programming, *Operations Research*, **14**, 568-594.

Lo, H.K. 1999. A novel traffic signal control formulation, *Transportation Research*, **33A** , 433-448.

Lo, H.K. 2001. A cell-based traffic control formulation: strategies and benefits of dynamic timing plans, *Transportation Science*, **35**(2), 148-164.

Lo, H.K., Chang, E., and Chan, Y.C. 2001. Dynamic network traffic control, *Transportation Research*, **35A**, 721-744.

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

Martin, P.T., Perrin, J., Chilukuri, B.R., Jhaveri, C., Feng, Y.Q. 2003. Mountain-Plains Consortium Report No. 03-141: Adaptive Signal Control II, University of Utah Traffic Lab, Salt Lake City, UT 84112

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

Mendel, J.M., McLaren, R.W., 1970. Reinforcement learning control and pattern recognition systems, *Adaptive, Learning, and Pattern Recognition Systems*: *Theory and Applications*, vol.**66**, J.M. Mendel and K.S. Fu, eds., 287-318, New York: Academic Press.

Mikami, S., Kakazu, Y. 1994. Genetic reinforcement learning for cooperative traffic signal control, in *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 223-228.

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

Nakamiti, G., Freitas, R. 2002. Adaptive, real-time traffic control management, *International Journal of Automotive Technology*, **3**(3), 89-94.

Nakamiti, G., Gomide, F. 1996. Fuzzy sets in distributed traffic control, *in Proc. 5th IEEE Int. Conf. Fuzzy Systems*, 1617-1623.

Nakatsuyama, M., Nagahashi, H., Nishizara, N. 1984. Fuzzy logic controller for a traffic junction in the one-way arterial road, *9th IFAC- World Congress*, Budapest, Hungary, 13-18.

Nedić, A., Bertsekas, D.P. 2003. Least squares policy evaluation algorithms with linear function approximation, *Discrete Event Dynamic Systems: Theory and Applications,* **13**, 79-110.

Papadaki, K., Powell, W. B., 2002. A monotone adaptive dynamic programming algorithm for a stochastic batch service problem, *European Journal of Operational Research*, **142**(1), 108-127.

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.

Papadaki, K., Powell, W. B., 2007. Monotonicity in multidimensional Markov decision processes for the batch dispatch problem, *Operations Research Letters*, **35**, 267-272.

Papis, C. and Mamdani, E. 1977. A fuzzy logic controller for a traffic junction*, IEEE Trans. Syst*., Man, and Cybern., **7**, 707-717.

Powell, W.B. 1996. A stochastic formulation of the dynamic assignment problem, with an application to truckload motor carriers, *Transportation Science*, **30**(3), 195-219.

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.

Powell, W.B., Topaloglu, H. 2003. Stochastic programming in transportations and logistics, in *Handbook in Operations Research and Management Science,* Volume on *Stochastic Programming*, Ruszczynski, A., Shapiro, A. (Eds.), North Holland, Amsterdam.

Powell, W.B., Van Roy, B. 2004. Approximate dynamic programming for high-dimensional resource allocation problems, in *Handbook of Learning and Approximate Dynamic Programming,* Jennie Si, J., Barto, A.G., Powell, W.B., Wunsch, D. (Eds.), Published by Wiley-IEEE, 2004, ISBN 047166054X.

PTV Planning Transport Verkehr AG. 2004 *User's Manual, VISSIM 4.0*, Karlsruhe, Germany

Puterman, M.L. 1994. Markov decision process: discrete stochastic dynamic programming, John Wiley & Sons, Inc., New York, ISBN 0-471-61977-9.

Richards, P.I., 1956. Shockwaves on the highway. *Operation Research,* **4**, 42-51.

Robertson, D.I. 1969. TRANSYT: a traffic network study tool. Report LR253, Road Research Laboratory, Ministry of Transport, Crowthorne, Berkshire.

Robertson, D.I., Bertherton, R.D., 1974. Optimum control of an intersection for any known sequence of vehicular arrivals, *Proceedings* of the 2nd IFAC-IFIP-IFORS Symposium on Traffic Control and Transportation system, Monte Carlo.

Roess, R.P., Prassas, E.S., McShane, W.R., 2004. Traffic Engineering, 3rd Edition, Upper Saddle River: Pearson Prentice Hall. ISBN 0-13-142471-8.

Singh, S.P., Jaakkola, T., Jordan, M.I. 1995. Reinforcement learning with soft state aggregation, in *Advances in Neural Information Processing Systems 7,* Tesauro, G., Touretzky, D.S., Leen, T.K. edits, MIT Press, Cambridge, MA.

Spall, J.C., Chin, D.C. 1997. Traffic-Responsive Signal Timing for system-wide traffic control, *Transportation Research Part C*, **5**, 153-163.

Srinivasan, D., Choy, M.C., Cheu, R.L. 2006. Neural networks for real-time traffic network control, *IEEE Trans. Intelligent Transport Systems*, **7**(3), 261-272.

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.

Texas Transportation Institute. 2007. The 2007 Annual Urban Mobility Report.

Tsitsiklis, J.N. 1994. Asynchronous stochastic approximation and Q-learning, *Machine Learning*, **16**, 185-202.

Tsitsiklis, J.N., Van Roy, B. 1996. Feature-based methods for large scale dynamic programming, *Machine Learning*, **22**, 59-94.

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. 1998. Learning and value function approximation in complex decision process, PhD thesis, MIT.

Van Zuylen, H.J. 1976. Acyclic traffic controllers. Note from the Centre for Transport Studies, UCL.

Venayagamoorthy, G.K., Harley, R.G., Wunsch II, D.C. 2000. Comparison of a heuristic dynamic programming and a dual heuristic programming based adaptive critics neurocontroller for a turbogenerator. IJCNN (3) 2000: 233-240.

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.

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**, Crowthorne: TRRL.

Watkins, C.J.C.H. 1989. Learning from Delayed Rewards, Thesis, University of Cambridge, England.

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

Webster, F.V. 1957. Traffic signal settings, *Road Research Technical Paper*, No.39, Road Research Laboratory, London.

Welch, P.D., 1981. On The problem of the initial transient in steady-state simulation, IBM Watson Research Centre, Yorktown Heights, N.Y.

Welch, P.D., 1983. The statistical analysis of simulation results. In Lavenberg, S.S. (Eds.) The Computer Performance Modelling Handbook, 268-328, Academic Press, N.Y.

Werbos, P.J. 1974. Beyond regression: New tools for prediction and analysis in the behavioral sciences, Ph.D Thesis, Harvard University, Cambridge, MA.

Werbos, P.J. 1990. Backpropagation through time: what it does and how to do it. *Proc. IEEE*, **78**(10), 1550-1560.

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.

Werbos, P.J. 2004. ADP: Goals, Opportunities and Principles, in *Handbook of Learning and Approximate Dynamic Programming,* Jennie Si, J., Barto, A.G., Powell, W.B., Wunsch, D. (Eds.), 3-44, Published by Wiley-IEEE, 2004, ISBN 047166054X.

Widrow, B., Hoff, M.E. 1960. Adaptive switching circuits, *IRE WESCON Convention Record,* 96-104.

Wong, C.K., Wong, S.C. 2003. Lane-based optimization of signal timings for isolated junctions, *Transportation Research*, **37B** (1), 63-84.

Wong, C.K., Wong, S.C., Lo, H.K. 2007. Reserve capacity of a signal-controlled network considering the effect of physical queuing, in *Proceedings of the 17th International Symposium on Transportation and Traffic Theory*, Edt. Allsop, R.E., Bell, M.G.H., Heydecker, B.G., Elsevier, 533-553.

Wood, K. 1993. Urban traffic control, systems review. Transport and Road Research Laboratory *Report* **PR 41**. Crowthorne: TRL.