Approximate Linear Programming for Traffic Control at Isolated Signalized Intersections

Angelique Mak

Vrije Universiteit
Faculty of Sciences
Business Mathematics and Informatics
De Boelelaan 1081a
1081 HV Amsterdam

November 2007

Preface

The BMI paper is one of the last compulsory subjects of the master study Business Mathematics and Informatics. The target of the BMI paper is describing and analyzing a problem in the field of BMI based on existing literature. Its focus embraces aspects of economics, mathematics, and computer science.

The building blocks of this paper are two papers. The first paper **[1]** describes an Approximate Linear Programming approach for average cost dynamic programming. A traffic control problem formulated in the second paper **[4]** is chosen to illustrate this approach. It focuses on the linear approach to approximate the dynamic programming value function through experiments with traffic control at isolated signalized intersections to find out how traffic light switching schemes for this system can be determined such that the number of cars in all flows is minimized in the long term.

I would like to thank my supervisor, Sandjai Bhulai. He provided me guidance and support, not only during my writing, but also most of the time during my master study BMI. Also his optimism has encouraged me and I am grateful for that.

Further, I would like to thank Paul Harkink, Chi Hung Mok, and Christel Nijman for their technical support. Without their generosity and help, finishing this thesis would have been much harder for me.

Management Summary in English

Intersections are places in the traffic network where many potential conflicts occur. Traffic is much affected by the traffic light control. A single intersection of two two-way streets with controllable traffic lights at each corner is considered in this paper. The main purpose of this paper is to apply the *two-phase Approximate Linear Programming approach for average cost dynamic programming* presented by De Farias and Van Roy **[1]**, to find out how traffic light switching schemes for this system can be determined such that the number of cars in all flows is minimized in the long term. Cars arrive at an intersection controlled by a traffic light and form a queue. The dynamic control of the traffic lights is based on the numbers of cars waiting in the queues. The model that describes the evolution of the queue lengths used in this paper is formulated in the paper of Haijema and Van der Wal **[4]**, which is modelled as a Markov Decision Process in discrete time. The set of all flows is partitioned into disjointed combinations of non-conflicting flows that will receive green together.

In principle, problems of this type can be solved via dynamic programming. Dynamic programming refers to a collection of algorithms that can be used to compute optimal policies given a model of the environment, such as a Markov Decision Process. Dynamic programming computes the optimal value function by solving the Bellman's equation. The domain of the optimal value function is the state space of the system to be controlled. This means that the number of variables (value function) to be stored is equal to the size of the state space. When the state space is large this dynamic programming computation becomes intractable. It is known as the 'curse of dimensionality'. Especially, when dealing with a multi-dimensional state space, its size grows exponentially in the number of state variables. This is also the case in traffic light control, due the fact that each queue (lane) has actually an infinite buffer.

Approximate dynamic programming intends to alleviate the curse of dimensionality by considering the approximation to the value function, the *scoring function*, which can be stored and computed efficiently. One of the considerations within Approximate Dynamic Programming is choosing the approximation architectures, the structure of the approximation to the value function. In this paper, the use of linear architectures is considered. A collection of functions that maps the system state space to real numbers (the basis functions) is chosen and the scoring function can be generated by finding an appropriate linear combination of these basis functions. Hence, it suffices to store the weights assigned to each of the basis functions in the linear combination instead of storing the value function for each state in the system. The number of variables (one per basis function) to be stored is tremendously smaller than the number used by the value function with one value per state in the system.

A successful use of approximate dynamic programming depends on a good choice of the basis functions and a good choice of weights assigned to each of the basis function in the linear combination. The dynamic programming problem can be recast as a linear programming problem. However, this exact linear programming approach also suffers from the curse of dimensionality. They have as many variables as the number of states in

the systems and at least the same number of constraints. Combining the exact linear programming approach with the linear approximation architecture leads to the *Approximate Linear Programming* algorithm (ALP). Compared to the exact linear program that stores the optimal value function for each state in the system, the ALP has a much smaller number of variables since that it has as many variables as the number of basis functions. There are two phases included in the ALP approach for average cost dynamic programming. The first phase prioritizes approximation of the optimal average cost, but does not necessarily give a good approximation to the value function. The second phase explicitly approximates the value function, with presence of the so called state relevance weights that is used for controlling the quality of the approximation to the optimal value function.

Based on the pre-specified basis functions and state relevance weights, it is observed that the ALP algorithm did a good job in approximating the dynamic programming value functions. It corresponds to the determination of the switching scheme for the traffic light control such that the number of cars in all flows is minimized in the long term.

Management Summary in Dutch

Kruispunten zijn plaatsen in het verkeersnetwerk waar veel potentiële conflicten voorkomen. Het verkeer wordt sterk beïnvloed door sturing van het verkeerslicht. Dit verslag behandelt een kruising van twee tweerichtingsstraten met controleerbare verkeerslichten op elke hoek van de straat. Het hoofddoel van dit verslag is om te weten hoe de omschakelingsregelingen van het verkeerslicht voor dit systeem kunnen worden bepaald, zodanig dat het aantal auto's in alle stromen op lange termijn wordt geminimaliseerd. Hiervoor wordt de voorgestelde benaderingsmethode door De Farias en Van Roy **[1]**, 'Two-phase Approximate Linear Programming for average cost dynamic programming' toegepast. De auto's komen bij een kruising aan die door een verkeerslicht wordt gestuurd en vormen een rij. De dynamische controle van de verkeerslichten is gebaseerd op het aantal auto's die in de rijen wachten. Het model dat de evolutie van de rijlengten beschrijft, is gebaseerd op de formulering van Haijema en Van der Wal **[4]**, dat gemodelleerd is als een Markov Decision Process in discrete tijd. De stromen worden verdeeld in disjuncte combinaties van conflictvrije stromen die samen groen licht krijgen.

In principe kan dit probleem worden opgelost via dynamische programmering. De dynamische programmering verwijst naar een verzameling van algoritmen die gebruikt kunnen worden om een optimaal beleid te vinden, gegeven een Markov Decision Process model. De dynamische programmering berekent de optimale *waardefunctie* door Bellman's vergelijking op te lossen. Het domein van de optimale waardefunctie is de *toestandsruimte* van het systeem. Dit zou betekenen dat het aantal variabelen (de opgeslagen waardefunctie) gelijk is aan de grootte van de toestandsruimte. De dynamische programmering is niet meer computationeel efficiënt wanneer de toestandsruimte van het systeem groot is. Dit probleem wordt meestal genoemd als de '*curse of dimensionality*'. Vooral, wanneer de toestandsruimte multidimensionaal is, groeit zijn grootte exponentieel in het aantal variabelen. Dit is ook het geval in het verkeerslichtprobleem, door het feit dat elke stroom een oneindige buffer heeft.

Approximate Dynamic Programming is bedoeld om het probleem van de 'curse of dimensionality' te verminderen door het benaderen van de waardefunctie, die efficiënt kan worden opgeslagen en verkregen. Eén van de overwegingen binnen Approximate Dynamic Programming is het kiezen van benaderingsstructuur, de structuur van de benaderende waardefunctie. In dit verslag wordt de lineaire architectuur gekozen. We kiezen een verzameling van functies (*basisfuncties*) die de *mapping* van toestandsruimte van het systeem naar de reële getallen geeft. De benadering van de waardefunctie (*scoring functie)* kan worden verkregen door een geschikte lineaire combinatie van de basisfuncties te vinden. Het is dus voldoende om de wegingsfactor van elke basisfunctie in de lineaire combinatie op te slaan in plaats van het opslaan van de waardefunctie voor elk toestand in het systeem. Het aantal variabelen is daardoor enorm kleiner ten opzichte van het aantal waardefuncties met één waarde per toestand in het systeem.

Een succesvol gebruik van *Approximate Dynamic Programming* hangt af van een goede keuze van de basisfuncties en een goede keuze van de wegingsfactor van elke basisfunctie in de lineaire combinatie. Het dynamische programmering probleem kan als lineaire

programmering (*LP)* probleem worden herschreven. Maar deze lineaire programmering benadering lijdt ook onder de zogenaamde 'curse of dimensionality'; ze hebben evenveel variabelen als het aantal toestanden in het systeem en minstens hetzelfde aantal restricties. Het combineren van de LP met de lineaire benaderingsarchitectuur leidt tot *Approximate Linear Programming* (ALP). De ALP heeft een veel kleiner aantal variabelen dan de LP omdat er zo veel variabelen zijn als het aantal gekozen basisfuncties. Er zijn twee fasen in de benadering van de ALP voor gemiddelde kosten dynamische programmering. De eerste fase is gericht op het benaderen van de optimale gemiddelde kosten, maar het geeft niet altijd een goede benadering voor de waardefunctie. De tweede fase benadert specifiek de waardefunctie.

Gebaseerd op de gekozen basisfuncties en *state relevance weights*, de ALP algoritme doet het goed in het benaderen van de waardefuncties. Dit zorgt voor het bepalen van de omschakelingsregelingen voor de verkeerscontrole probleem dusdanig dat het aantal auto's in alle stromen op lange termijn geminimaliseerd wordt.

Contents

1 Introduction

As the number of road users and the need of transportation increases, cities around the world face serious road traffic congestion problems. Traffic jams have become the everyday life's ritual for most of the people in the world. Traffic jams do not only cause tremendous costs due to unproductive time losses; they also cause the increasing probability of accidents and have a negative impact on the environment (congestion wastes fuel and increases air pollution due to increased idling, acceleration, and braking) and on the quality of life (stress and frustration). This leads to the question about the possibility of controlling traffic flows in order to reduce the traffic jams.

Intersections are places in the traffic network where many conflicts can occur potentially. These conflicts exist because an intersection is a road area where multiple traffic flows meet or cross. Reducing conflicts can be accomplished through a combination of efforts, including the careful use of the road infrastructure, comprehensive traffic safety laws and regulations, sustained education of drivers, the willingness among drivers to obey the traffic safety laws, and traffic management. The traffic management around the intersection area is done by the traffic light control.

In most countries, three-state traffic light is used **[6]**. The sequence is red, green and yellow which means stop, go and prepare to stop, respectively. The most control strategies found in practice are cyclic; the order in which the groups of flows are served is fixed. There is a range of several logical policies by which the traffic light can be controlled. The most basic policy can be classified according to the following characteristics **[4]**:

Fixed-time (FC) control. In this form of operation, not only the order is fixed but the red, yellow, and green light indications are timed at fixed intervals. Fixed cycle controllers are best suited for intersections where traffic volumes are predictable, stable, and fairly constant.

Unlike the FC, intersections with *traffic-responsive* control consist of actuated traffic controllers and vehicle detectors placed on the lanes approaching the intersection. This form of control makes use of real-time measurements. The length of the green interval can be lengthened or shortened based on the present volume of the traffic. At a hectic intersection, the green interval would be lengthened, or one gets a green period on inquiry.

Under *exhaustive* (XH) control**,** the green signals will be kept until all flows that have right of way are 'exhausted' (empty). The cyclic variant of exhaustive control is abbreviated by XHC. The alternative form of exhaustive control is *anticipative exhaustive* control XHC(1) and XHC(2), which anticipates departures during 1 and 2 yellow slots, respectively. In other words, the green periods will be kept until the number of cars at each flow in the combination that has right of way is at most one and two in XHC(1) and XHC(2), respectively.

Isolated control is applicable to single intersections. The signals are operated without consideration of any adjacent signals. In such a case, each intersection will have a signal control that is most appropriate for that single intersection. On the contrary, *coordinated control* considers an urban zone or even a whole network comprising many intersections.

However, the annual toll of accidents due to motor vehicle crashes has not substantially changed in more than 25 years despite improved intersection infrastructures and more sophisticated application of traffic engineering measures. As mentioned in **[7],** installing signals do not always make intersections safer. The installation of signals that operate improperly can create situations where overall intersection congestion is increased, which in turn can create aggressive driving behavior. Drivers tend to become impatient and violate red lights when the traffic light control causes longer waiting times at intersections. This subjects local residents to a greater risk of collisions, worse congestions and more air and noise pollution. Hence, appropriate traffic control decisions are required.

This paper focuses on the dynamic control at a signalized single intersection of two two-way streets with isolated control. This control problem can be formulated as a Markov Decision Process (MDP) for a stochastic dynamical system with the average cost criterion. The state of system evolves under uncertainty and a sequence of decisions has to be made. Based on the real-time situation, i.e., the number of cars waiting in the queues, a decision has to be made as to which set of flows has right of way and these decisions have a long term effect. The current action determines a new configuration that determines which configurations may be reached in the future. A solution to a Markov decision problem is a *policy* (a mapping from a state to an action) that determines state transitions to minimize the average cost, which is the long-run number of cars in all flows. To be able to derive the optimal policies, a function defined on the state space called the value function is required to be computed and stored. For most problems of practical interest, the state space is extremely large so that computing and storing the optimal value function require a lot of time. This large space makes the dynamic programming computationally intractable.

Because of the computational complexity of solving the dynamic program, much effort has been put into finding alternative learning algorithms. For instance, Haijema and Van der Wal **[4]** presented an approach to smoothen the traffic flow that starts from a (nearly) optimal fixed cycle strategy and executes one policy improvement step that leads to a dynamic control strategy.

Another approach is Approximate Dynamic Programming (ADP). This approach is about finding a good approximation to the value function. This paper applies the linear programming approach to approximate dynamic programming proposed by De Farias and Van Roy **[1]** for solving the traffic control problem that is formulated as a discrete time MDP. This approach considers the average cost criterion and a version of the approximate linear program that generates approximations to the optimal average cost and value function.

2 Approximate Dynamic Programming

Markov decision processes (MDP's) provide a mathematical framework for modelling decision making in situations under uncertainty. Given a model of the environment as a Markov decision process, dynamic programming can be used to compute the optimal policies. This chapter gives a description of how dynamic programming offers a solution to the problem of minimizing the average cost over a finite horizon. Moreover, we discuss how the curse of dimensionality affects the dynamic programming algorithm. Further, the main ideas in approximate dynamic programming will be presented.

2.1 Markov Decision Processes

Dynamic programming offers a solution to problems involving sequential decision making in systems with non-linear, stochastic dynamics. Systems in this setting are described by a set of variables evolving over time – the *state variables*. The state variables take values in the *state space* of the system, which is the set of all possible *states* the system can be in. The main idea in dynamic programming is that an optimal decision can be derived based on the score assigned to each of the states in the system – the *value function*. The optimal value function obtained from dynamic programming captures the advantage of being in a given state relative to being in all other states.

Consider discrete-time stochastic control problems involving a finite state space S of cardinality $|S| = N$. For each state $x \in S$, there are possible actions that can be chosen $\mathcal{A}_x$. In a given state x when action a is taken, a cost of $c(x,a)$ is incurred. The transition probabilities $p(x,a,y)$, for each state pair (x,y) and action $a \in \mathcal{A}_x$, represent the probability that given action a while being in state x, the next state will be y.

A *policy* is a mapping from a state to an action. Under policy u, the system follows a Markov process with transition probabilities $p_u(x,y)$.

The solution to a Markov Decision Process can be expressed as a policy u, which gives the action to take for a given state, regardless of the prior history. Let x_t denote the random variable for the state that the system is in at time t and $c_u(x_t)$ denotes the corresponding cost when policy u is taken. Then, it is well known that there exists a policy u such that $\frac{1}{T} E\left[\sum_{t=0}^{T-1} c_u(x_t) \mid x_0 = x\right]$, as T goes to infinity, is minimized simultaneously for all states and the aim is to identify that policy.

Here, the Markov process is assumed to be irreducible; for each pair of state (x,y) and each policy u, there is a t such that $P_u^t(x,y) > 0$. In other words, it is possible to get to any state from any state. This implies that, for each policy u, the limit

$\lim_{T\to\infty}\frac{1}{T}E\left[\sum_{t=0}^{T-1}c_u(x_t)\mid x_0=x\right]$ exists and the average cost is independent of the initial state in the system.

Denote the optimal average cost by $g^* = \min_u g_u$. The optimal policy with the average cost criterion can be derived from the solution of *Bellman's equation*,

$$g+V(x)=\min_{a\in A_x}\left\{c(x,a)+\sum_y p(x,a,y)V(y)\right\}. \qquad \textbf{(2-1)}$$

where $V(\cdot)$ denote value function. The interpretation of $V(x)$ is the difference in accrued cost when starting the process in state x relative to a reference state. Bellman's equation can be formulated in terms of matrices as follows.

$$g.e+V=c_{u^*}+P_{u^*}V\ , \qquad \textbf{(2-2)}$$

where e is a vector with 1 as entries, c_{u^*} and P_{u^*} are vectors of the costs and the transition probabilities based on the optimal policy, respectively.

Denote the solution of Bellman's equation by pairs (g^*,V^*). An alternative method for deriving V is so called policy iteration. This algorithm starts with considering a policy π. The corresponding (g,V) can be obtained by solving the Bellman's equation:

$$\pi(x)=c(x,\pi(x))+\sum_y p(x,\pi(x),y)V(y). \qquad \textbf{(2-3)}$$

To improve the policy π, take

$$\pi'(x)=\arg\min_{a\in A}\left\{c(x,a)+\sum_y p(x,a,y)V(y)\right\} \qquad \textbf{(2-4)}$$

in each state. The corresponding (g',V') can be obtained by again solving the Bellman's equation. The improvement can be again obtained by solving **(2-4)** based on V'. This iteration will be continued until the minimum is attained for each state. Note that the value function for every state has to be stored in memory in every step. Therefore, the applicability of dynamic programming is severely limited. The domain of the optimal value function is the state space of the system to be controlled. This means that the number of variables (value function) to be stored and computed is equal to the size of the state space. When the state space is large this dynamic programming method becomes computationally intractable. Especially when dealing with multi-dimensional state space, its size grows exponentially in the number of state variables. This problem is called curse of dimensionality.

2.2 ADP with a linear approximation architecture

To alleviate the curse of dimensionality, the problem is solved by finding an approximation to the value function, $\tilde{V} : S \times \Re^K \mapsto \Re$, called the *scoring function*. The underlying assumption is that the value function has some structure such that a reasonable approximation exists.

By using the linear approximation architecture, the scoring function is generated within a parameterized class of functions. It maps the system state space to the set of real numbers. Consider a given set of *basis functions* $\phi_i : S \mapsto \Re, \quad i = 1, \ldots, K$, the scoring functions are represented as linear combinations of the basis functions:

$$\tilde{V}(\cdot, r) = \sum_{i=1}^{K} \phi_i(x) r_i. \qquad \textbf{(2-5)}$$

Imagine that the pre-selected basis functions are stored as columns of matrix $\Phi \in \Re^{S \times K}$, and each row corresponds to the basis functions evaluated at a different state x.

$$\Phi = \begin{bmatrix} | & & | \\ \phi_1 & \vdots & \phi_K \\ | & & | \end{bmatrix}. \qquad \textbf{(2-6)}$$

Now the optimization problem is formulated and analyzed as an optimization problem for computing the weights $r_i \in \Re^K$. Hence, it suffices to store the weights assigned to each of the basis functions in the linear combination instead of storing the value function for each state in the system. The number of variables (one per basis function) to be stored is tremendously smaller than the number compared to the value function with one value per state in the system.

3 Approximate Linear Programming for average costs

A successful use of approximate dynamic programming depends on a good choice of the basis functions and a good choice of weights assigned to each of the basis functions in the linear combination. A study to the optimal selection of basis functions is out of the scope of this paper. Therefore, we assume that the set of basis functions is pre-specified, and that the focus is on finding an appropriate parameter vector $r \in \Re^K$, given a pre-selected set of basis functions.

It is known that the dynamic programming problem can be recast as a linear programming problem. However, this exact linear programming approach also suffers from the curse of dimensionality. They have as many variables as the number of states in the system and at least the same number of constraints. Combining the exact linear programming approach with the linear approximation architecture leads to the *approximate linear programming* algorithm (ALP). Compared to the exact linear program that stores the optimal value function for each state in the system, the ALP has a much smaller number of variables since it has as many variables as the number of basis functions. In the next sections, the two-phase ALP approach for average costs is described. The first phase of the average-cost ALP prioritizes approximation of the optimal average cost, but does not necessarily give a good approximation to the value function. The second phase explicitly approximates the value function.

3.1 First phase of the average-cost ALP

Recall the Bellman's equation (see Equation **(2-1)**). It can be solved by the average cost Exact Linear Programming (ELP):

$$\begin{aligned} \max_{g,V} \quad & g \\ s.t. \quad & g + V(x) \le \min_{a \in A_x} \left\{ c(x,a) + \sum_y p(x,a,y) V(y) \right\}, \quad \forall x. \end{aligned} \tag{3-1}$$

The problem is translated in a maximization of the average cost that would be subject to inequalities of the form "≤" which corresponds to upper bounds. Note that the constraints are non-linear, each constraint involves a minimization over the possible actions. But each constraint can be decomposed into $|\mathcal{A}_x|$ constraints. Therefore, problem **(3-1)** can be seen as a Linear Programming described by **(3-2)**.

$$\begin{aligned} \max_{g,V} \quad & g \\ s.t. \quad & g + V(x) \le c(x,a) + \sum_y p(x,a,y) V(y), \quad \forall x, a. \end{aligned} \tag{3-2}$$

This results in a total of $|S| \times |\mathcal{A}_x| + 1$ constraints, which is unmanageable if the state space is large. The combination of the exact linear programming and the linear approximation architecture leads to the *first phase ALP* described by **(3-3)**.

$$\begin{aligned} \max_{g,r} \quad & g \\ s.t. \quad & g + \Phi r(x) \le c(x,a) + \sum_y p(x,a,y)\Phi r(y), \quad \forall x, a. \end{aligned} \tag{3-3}$$

Denote the solution of the first phase ALP by (g_1, r_1). Note that the maximization problem in **(3-3)** is equivalent to minimizing $|g^* - g_1|$. Since the first phase ALP corresponds to the exact LP **(3-1)** with the extra constraint $V = \Phi r$, the solution to the first phase ALP is limited, $g_1 \le g^*$ for all feasible g_1. This implies that the first phase ALP can be seen as an algorithm to approximate the optimal average cost.

Compared with the ELP that stores the optimal value function for each state in the system, the ALP has a much smaller number of variables since that it has as many variables as the number of basis functions plus one. However, the ALP has still as many constraints as the number of state-action pairs.

3.2 Second phase of the average-cost ALP

It turns out, from an example given in the paper **[1]**, that even though the first phase ALP produces a good approximation to the optimal average cost, it can produce arbitrarily bad policies. The main problem is that the algorithm of the first phase ALP has priority to approximate the optimal average cost, but it does not necessarily yield a good approximation to the optimal value function. Hence, a two-phase average-cost ALP is proposed in which the first phase is simply the first phase of the average-cost ALP introduced in Section 3.1. In the first phase, the approximation to the optimal average cost is generated, while in the second phase the focus is on the approximation to the optimal value function.

The second phase of the ALP is formulated as follows.

$$\begin{aligned} \max_{r} \quad & c^T \Phi r \\ s.t. \quad & g_2 + \Phi r(x) \le c(x,a) + \sum_y p(x,a,y)\Phi r(y), \quad \forall x \ne 0, a. \end{aligned} \tag{3-4}$$

The parameters that have to be pre-specified are the state relevance weights $c > 0$ and g_2. Denote the optimal solution of the second phase of the average-cost ALP by r_2. In De Farias and Van der Roy **[1]**, a lemma and some theorems were described and used to understand how the state relevance weights c and the estimated optimal average cost g_2 in the second phase of the ALP can be used for controlling the quality of the approximation to the optimal value function.

In the following theorem, the interpretation of second phase ALP as the minimization of a certain weighted norm of the approximation error is given, with weights equal to the state relevance weights.

Theorem 1 (De Farias and Van Roy [1]):

Let r_2 be the optimal solution to the two-phase ALP. It minimizes $\|V_{g_2} - \Phi r\|_{1,c}$ over the feasible region of the two-phase ALP.
Proof: The norm $\|.\|_{1,c}$ is defined by $\|V\|_{1,c} = \sum_{x \in S} c(x)\,|V(x)|$.
Maximizing $c^T \Phi r$ is equivalent to minimizing $c^T (V_{g_2} - \Phi r)$. It is well known that for all V, $(I - P_{\pi^*})^{-1}(c_{\pi^*} - g_2 e) \geq V$, we have $V \leq V_{g_2}$. Hence, any r that is a feasible solution to two-phase ALP problems satisfies $\Phi r \leq V_{g_2}$. It follows that
$$\|V_{g_2} - \Phi r\|_{1,c} = \sum_{x \in S} c(x)\,|V_{g_2}(x) - \Phi r(x)| = c^T V_{g_2} - c^T \Phi r,$$
and maximizing $c^T \Phi r$ is therefore equivalent to minimizing $\|V_{g_2} - \Phi r\|_{1,c}$.

Hence, any fixed choice of g_2, that satisfies $g_2 \leq g^*$, there is bound

$$\|V^* - \Phi r_2\|_{1,c} \leq \|V_{g_2} - \Phi r_2\|_{1,c} + (g^* - g_2) c^T (I - P_{\pi^*})^{-1} e. \qquad \textbf{(3-5)}$$

The two-phase ALP minimizes the upper bound on the norm $\|V^* - \Phi r_2\|_{1,c}$ of the error in the approximated value function. The state relevance weight c determines how errors over different regions of the state space are weighted when approximating the optimal value functions, and can be used for specifying the trade-off in the quality of the approximation across different states. Therefore, to generate a better approximation in a region of the state space one can assign relatively larger weights to that region. To have some clue on how to choose the appropriate state relevance weights *c*, performance bounds will be provided in the next section.

3.3 State Relevance Weights

A bound on the performance of greedy policies associated with approximate value functions were presented in De Farias and Van Roy **[1]** that provides some guidance on choosing appropriate state relevance weights. The bound is described in Theorem 2.

Theorem 2 (De Farias and Van Roy [1]):

For all V, let g_V and π_V denote the average cost and the stationary state distribution of the greedy policy associated with V. Then, for all V such that $V \leq V^*$,
$$g_V \leq g^* + \|V^* - V\|_{1,\pi_V}.$$
Proof:

Note that the average cost associated with V is given by $g_V = \pi_V^T c_V$ and $\pi_V^T P_V = \pi_V^T$ is valid for the stationary state distribution. c_V and P_V denote the costs associated with the greedy policy with respect to V. g_V can be formulated as $g_V = \pi_V^T c_V = \pi_V^T (c_V + P_V V - V)$. Now if $V \le V^*$, then

$$\begin{aligned}\pi_V^T (c_V + P_V V - V) &\le \pi_V^T (c_{V^*} + P_{V^*} V^* - V) \\ &= \pi_V^T (g^* .e + V^* - V) \\ &= g^* .e + \pi_V^T (V^* - V) = g^* .e + \left\| V^* - V \right\|_{1,\pi_V}\end{aligned}$$

The performance bound described in Theorem 2 gives an alternative for selecting state relevance weights. One approach is to select the state relevance weight corresponding to the stationary state distribution associated with the greedy policy. It seems logical, since the aim is to have a good approximation to the value function, importantly, thus the states that are visited more often need to be approximated better. One difficulty with obtaining the stationary state distribution is that one should know the optimal policy beforehand and the problem is finding the optimal policy yet. It suggests an iterative scheme using in each iteration the weights corresponding to the stationary state distribution associated with the policy generated by the previous iteration.

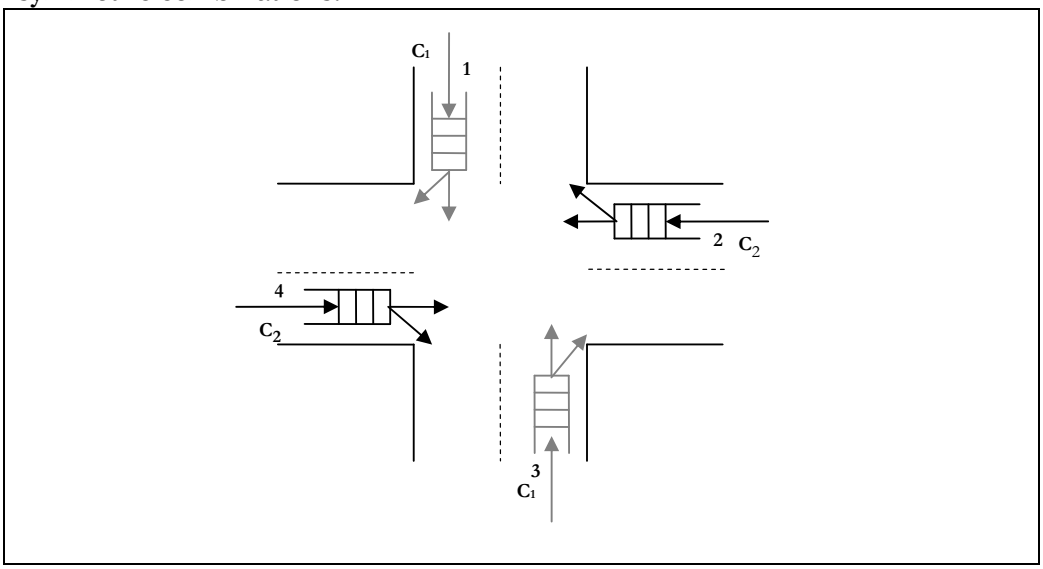
4 Traffic light control

The aim of this paper is to solve the traffic light control problem with the ALP approach described in Chapter 3. In this chapter, the basic notion and the modelling assumptions will be introduced. Furthermore, the problem will be formulated as a Markov Decision Problem.

4.1 Basic notation and the modelling assumptions

Consider a simple intersection of two two-way streets, F4C2, which is illustrated in Figure 4.1. The flows are numbered clockwise. Cars that arrive at one of the lanes go either straight crossing the intersection or make a left turn. The set of 4 flows is partitioned into 2 disjoint subsets, C_1 and C_2. A subset of flows is called a *combination*. Two compatible flows can safely cross the intersection simultaneously, else they are called *antagonistic*. The combinations are fixed, and they are chosen such that there is a conflict-free intersection. The flows 1 and 3 are considered as C_1 and the flows 2 and 4 constitute C_2. Flows in the same combination will always have the same light indication at the same time. When one combination has green or yellow indication, another combination has red indication.

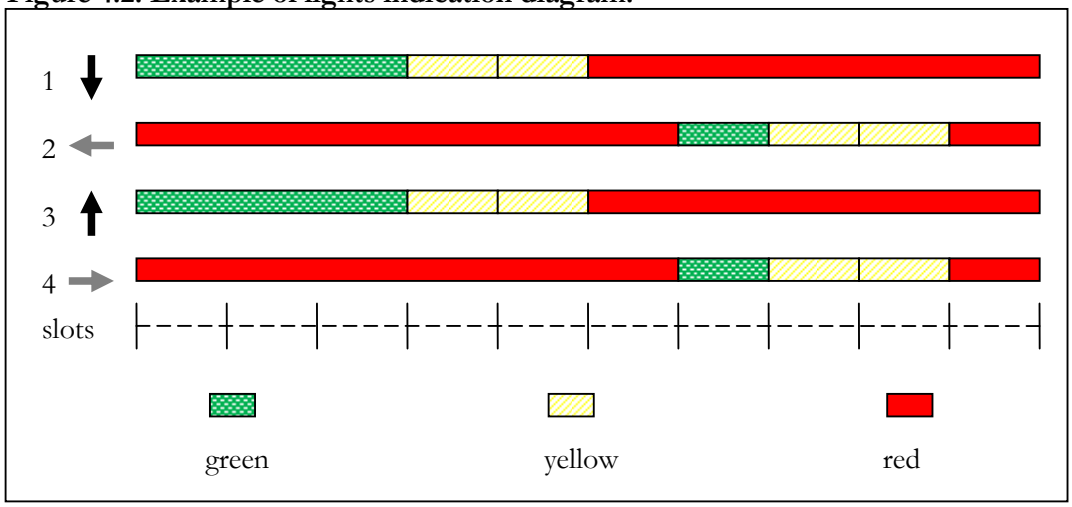
Figure 4.1: Two two-way streets intersection, F4C2, which serve 4 flows in 2 symmetric combinations.



For the sake of simplicity, the problem is formulated in discrete time. Time is divided into slots. This time unit *(slot)* is taken to be the time a car needs to cross the intersection when the light is green or yellow. Haijema and Van der Wal **[4]** assumed this time unit as being two seconds.

To avoid interference between antagonistic streams of consecutive slots when switching from a green indication for one combination to a green for a different combination, a *switching time* is necessary. The switching time is chosen to be fixed and it takes 3 slots; 2 slots of yellow and 1 slot in which all flows have a red indication. As an example, a cycle for the light indication for the flows is shown in Figure 4.2. The flows 1 and 3 get a green indication during 3 slots and the flows 2 and 4 get a green indication during 1 slot. The switching time takes 3 slots. Hence, the duration of the cycle is 10 slots.

Figure 4.2: Example of lights indication diagram.



The arrivals in different flows and in different slots are independent. It is reasonable to assume that the number of car arrivals in one slot is either 0 or 1 per flow. We denote the *arrival rate* in one slot by q_f for flow f.

In each flow that has right of way (having a green or a yellow indication), exactly one car can pass the stopping line in one slot. A car that arrives at an empty queue that has right of way passes the stopping line without delay.

The state of the process is observed in each slot. The decision epochs are as follows. New arrivals take place at the beginning of the slot, which is after the observation of the state of the process. Departures take place at the end of the slot prior to the observation of the new state. Hence when a flow has right of way and a car arrives in the certain slot, the state of the flow remains the same for the next slot.

4.2 Markov decision problem formulation

As stated above, the problem is considered as a discrete-time stochastic control problem. The Markov decision problem formulation consists of the specification of the states space, the decision space (in each state there are several actions from which the decision must be chosen), the transition probabilities, and the cost function.

4.2.1 States

The state of the system is represented by two vectors, one represents the state of the traffic and the other represents the state of the lights. The state of the traffic is fully described by a vector $\underline{k} = (k_1, k_2, k_3, k_4)$, with k_f the number of cars in flow f present at the beginning of a slot. Further, the state of the light is described by vector $x = (l, i)$ with $l \in \{1,2\}$ the combination which is having a green $(i = 0)$, a first yellow $(i = 1)$, a second yellow $(i = 2)$, or a red $(i = 3)$ light. The state of the light is fully described by x, because when one combination of the flows has right of way (having a green or a yellow indication), the other flows all have a red indication. Hence, the states are denoted by the vector $(\underline{k}, \underline{x}) = ((k_1, k_2, k_3, k_4), (l, i))$, in total a 6-dimensional vector.

4.2.2 Decisions

In each state there are several actions from which decision must be chosen. The decisions depend on the state of the traffic lights but also on the lengths of the queues. The possible decisions in the various situations are described as follows.

- If all lights are red, the possible decisions are to keep all lights red, or to give a green indication to one of the combinations.
- If the lights are green for one combination, there are two possible decisions: keep the lights as they are or change from green to first yellow.
- At the end of a first yellow slot there is only one decision: continue to the second yellow slot.
- After the second yellow, the only decision is to change into red for all flows.

Hence the decision space, denoted by $\mathcal{A}(\underline{k}, (l, i))$, is

$$\mathcal{A}(\underline{k}, (l, i)) = \begin{cases} \{(l,0), (l,1)\}, & if \quad i = 0 \\ (l, i+1), & if \quad i = 1,2 \\ \{(l,3), (l',0)\}, & if \quad i = 3 \end{cases} , \qquad \textbf{(4-1)}$$

with l' the next non-empty combination. Decisions are taken at the beginning of a slot and executed instantaneously. Thus, if a combination has right of way, cars of that combination can leave in the very same slot.

4.2.3 Transition probabilities

Given a state $(\underline{k}, \underline{x})$, the chosen action $\underline{a}$ implies an instantaneous change of lights from state $\underline{x}$ into state $\underline{a}$, due to the fact that the chosen action is part of the state. Hence, the transition probability from state $(\underline{k}, \underline{x})$ to state $(\underline{k}', \underline{x}')$ is 0 unless $\underline{x}' = \underline{a}$. The transition probabilities, denoted by $p(\underline{k}, \underline{x}; \underline{k}', \underline{a})$, are best described by considering each flow separately. Let $p_f(k_f, \underline{a}, k_f')$ denote the transition probability for the number of cars in flow f when action $\underline{a}$ is taken. Since the flows are independent, the transition probabilities are simply the product of transition probabilities for each flow.

$$p_f\left(\underline{k},\underline{x},\underline{k}',\underline{a}\right)=\prod_{f=1}^{4} p_f\left(k_f,\underline{a},k_f'\right). \qquad \textbf{(4-2)}$$

If by action $\underline{a}$ flow f has right of way during the coming slot, the transition probabilities per flow are given by

$$\begin{aligned} &p_f\left(k_f,\underline{a},k_f-1\right)=1-q_f; \quad p\left(k_f,\underline{a},k_f\right)=q_f, \quad k_f>0 \\ &p_f\left(0,\underline{a},0\right)=1. \end{aligned} \qquad \textbf{(4-3)}$$

And if action $\underline{a}$ implies red for flow f, then

$$p_f\left(k_f,\underline{a},k_f\right)=1-q_f; \quad p_f\left(k_f,\underline{a},k_f+1\right)=q_f, \quad k\geq 0. \qquad \textbf{(4-4)}$$

The transition probabilities for the number of cars in flow f are illustrated in Figure 4.3 and Figure 4.4.

Figure 4.3: Illustration of the transition probability for the number of cars in flow *f* when flow *f* has right of way.

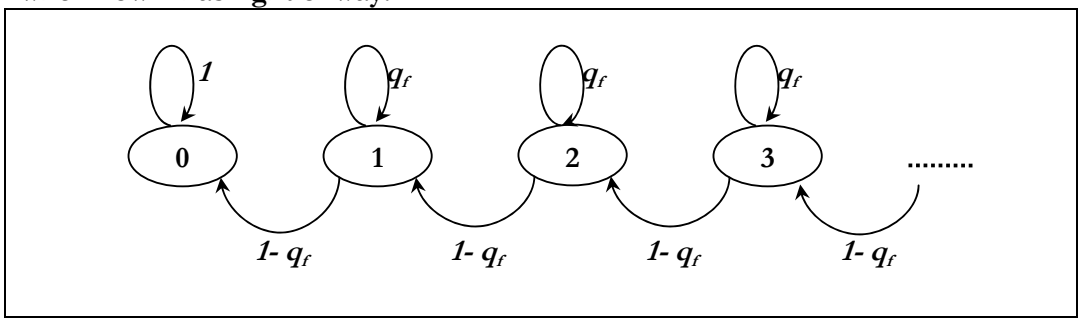
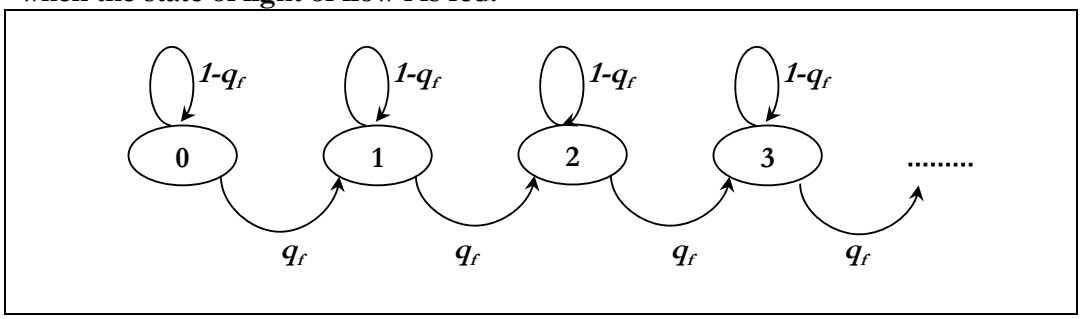


Figure 4.4: Illustration of the transition probability for the number of cars in flow *f* when the state of light of flow *f* is red.



4.2.4 Costs

The aim is to minimize the overall average waiting time per car. Based on *Little's Law*, this corresponds to minimizing the average number of cars waiting at the queues.

Therefore, a linear cost function is considered where one unit of costs is accounted for every car present at the beginning of a slot. Hence, the cost function, denoted by $c(\underline{k})$, is given by

$$c(\underline{k}) = \sum_{f=1}^{4} k_f \ . \qquad \textbf{(4-5)}$$

4.2.5 Countable state spaces

To easily compute the performance measures (and the optimal policy), the state space has to be countable. The state space can be reduced to be a finite one by limiting the number of cars that can be present in each flow. The maximum state N in each flow becomes one of the parameters in the system. Thus, the arrivals to a queue which is full will be rejected. This stochastic control problem involves a finite state space S of cardinality $|S| = N^4 \times 2 \times 4$, where the state of the system is a 6-dimensional vector. The transition probabilities should be changed with respect to q_f such that there are no transitions from state N to $N+1$. The transition probabilities per flow in the finite state space are defined as follows. If during the coming slot flow f has right of way, the transition probability is defined as follows,

$$\begin{aligned} &p_f(k_f, \underline{a}, k_f - 1) = 1 - q_f; \quad p(k_f, \underline{a}, k_f) = q_f, \quad 0 < k_f \le N, \\ &p_f(0, \underline{a}, 0) = 1. \end{aligned} \qquad \textbf{(4-6)}$$

If during the coming slot flow f gets a red indication, the transition probability in the case of countable state spaces is defined as follow,

$$\begin{aligned} &p_f(k_f, \underline{a}, k_f) = 1 - q_f; \quad p_f(k_f, \underline{a}, k_f + 1) = q_f, \quad k_f < N, \\ &p_f(N, \underline{a}, N) = 1. \end{aligned} \qquad \textbf{(4-7)}$$

5 The two-phase ALP approach for F4C2

After defining the model, the ALP formulation for this problem can be determined. The control strategy used in this paper is cyclic. The implementation of the approach is done in the following steps.

1. Find the optimal average cost based on the first phase of the two-phase ALP, denoted by g_1. The greedy policies π_1 associated with g_1 can be obtained. The first phase of the two-phase ALP is done given the predefined basis functions.
2. Evaluate the policies π_1 by simulating the state of each flow for *B* slots, say 50000, and determine the state of the lights based on the greedy policies π_1. Then, the average cost can be computed by taking the average of the sum of the number of cars in all flows, denoted by g_{1*}. It is expected that $g_{1*} \geq g_1$ because the first phase of the two-phase ALP does not necessarily give a good approximation to the value function which leads to non-optimal policies π_1.
3. Improve the policies π_1 by using the second phase of the two-phase ALP approach, given the approximation of the optimal average cost g_1, the predefined basis functions, and the state relevance weights. The new policies obtained will be denoted by π_2. The average cost obtained will be denoted by g_{2*}.
4. Evaluate the new policies π_2 by simulation as in step 2. It is expected that $g_{1*} \geq g_{2*} \geq g_1$.

The steps are further discussed in the next section. The arrival rates will be varied to compare the approach with the other strategies. The results obtained are reported in Section 5.2.

5.1 The two-phase ALP formulation

To ensure uniqueness, assume state V($\underline{0}$,(1,0)) acts as a reference state by taking V($\underline{0}$,(1,0))=0. The first-phase ALP is given by

$$\begin{aligned} \max_{g,r} \quad & g \\ s.t. \quad & g + \Phi r(\underline{k},\underline{x}) \leq c(\underline{k}) + \sum_{y} p(\underline{k},\underline{x},\underline{k}',\underline{a}) \Phi r(\underline{k}',\underline{a}), \quad \forall (\underline{k},\underline{x}), \underline{a}. \end{aligned} \tag{5-1}$$

Denote the solutions by (g_1, r_1). The basis functions that are used are

$$\begin{aligned} & \phi_0 = 1, \\ & \phi_a = k_a, \\ & \phi_{ab} = k_a k_b; \quad a,b \in \{1,2,3,4\}, \end{aligned} \tag{5-2}$$

where k_b is the number of cars in flow *b*. The value function is approximated by a second order polynomial that includes terms that correlate different flows with each other, with different weights for each state of light (l,i). Hence, the value function is $V(\underline{k},(l.i))$ approximated by $\Phi r(\underline{k},(l.i))$, which is

$$\Phi r(\underline{k},(l,i)) = r_0^{(l,i)} + r_1^{(l,i)}k_1 + \ldots + r_4^{(l,i)}k_4 + r_{11}^{(l,i)}k_1^2 + \ldots + r_{44}^{(l,i)}k_4^2 + r_{12}^{(l,i)}k_1k_2 + r_{13}^{(l,i)}k_1k_3 + r_{14}^{(l,i)}k_1k_4 + r_{23}^{(l,i)}k_2k_3 + r_{24}^{(l,i)}k_2k_4 + r_{34}^{(l,i)}k_3k_4. \quad \textbf{(5-3)}$$

The constant term $r_0^{(1,0)}$ =0, since $V(\underline{0},(1.0)) = 0$ is chosen. There are in total 15×8 = **120** variables. In this case, the number of variables does not change as the number of states grows. The linear programming is solved by using the *Premium Solver Platform for Excel,* which is able to solve linear programming problems with a number of variables up to 8,000. While the ALP may involve manageable number of variables, the number of constraints is still as many as the number of state-action pairs. Therefore, to be able to solve the ALP problem, the largest state on each flow has to be limited to be able to include all the state-action constraints for solving the LP problem. The maximum of the largest state on each flow is 4 corresponding to 7,500 constraints based on the MDP formulation. The optimal policy π_1 associated with g_1 can be generated by taking

$$\pi_1(\underline{k},\underline{x}) = \underset{\underline{a}\in A(\underline{k},(l,i))}{\arg\min}\left\{\sum_{k'} p(\underline{k},\underline{x};\underline{k}',\underline{a})\Phi r_1(\underline{k}',\underline{a})\right\}. \quad \textbf{(5-4)}$$

In order to check whether the first phase of the two-phase ALP yields a good approximation for the value function, simulation of the states will be done for 500,000 slots. The second phase of the two-phase ALP is done by solving linear problem **(5-5)**.

$$\begin{aligned} \max_{r} \quad & c^T\Phi r \\ s.t. \quad & g_2 + \Phi r(\underline{k},\underline{x}) \le c(\underline{k}) + \sum_{y} p(\underline{k},\underline{x},\underline{k}',\underline{a})\Phi r(\underline{k}',\underline{a}), \quad \forall(\underline{k},\underline{x}),a \end{aligned} \quad \textbf{(5-5)}$$

An obvious choice for g_2 is g_1, the estimate for the optimal average cost obtained from the first phase ALP. The solution to the second phase ALP is r_2. It is aimed to control the accuracy of the approximation to the cost function over different portions of the state space. As described in Theorem 1, maximizing $c^T\Phi r$ is equivalent to minimizing $c^T(V_{g_2} - \Phi r)$, which is the sum the errors of the approximation to the cost function weighted by c for balancing accuracy of the approximation over different states. Based on Theorem 2, the state relevance weights can be chosen corresponding to the stationary state distribution. It may suffice to use rough guesses about the stationary state distribution in some cases. Thus, the state relevance weights c are chosen in the form

$$c(\underline{k},(l,i)) = \begin{cases} \frac{1}{a}(1-\rho_i)^2\rho_i^{(k_1+k_3)}(1-\rho_3)^2\rho_3^{(k_2+k_4)}, & if \quad l=1, \\ \frac{1}{a}(1-\rho_i)^2\rho_i^{(k_2+k_4)}(1-\rho_3)^2\rho_3^{(k_1+k_3)}, & if \quad l=2, \end{cases} \quad \textbf{(5-6)}$$

where

$$a = \sum_{\underline{k},(l,i)} c(\underline{k},(l,i)). \quad \textbf{(5-7)}$$

To make sure that the sum of state relevance weights is 1, $c(\underline{k},(l,i))$ is multiplied by $\frac{1}{a}$.

5.2 Results and evaluation

5.2.1 Symmetric arrival rates

In this section, the results for varying arrival rates at a symmetric F4C2 intersection are presented in which all flows have identical arrival rates per slot. The average cost g_1 resulted by the first phase ALP and the average cost g_{1*} and g_{2*} yielded by the greedy policy with respect to Φr_1 and Φr_2, respectively, are given in the Table 5-1 below. The second phase ALP is done for the state relevance weights c with ρ_i =0.99 for all *i*. As observed, the results from the second phase ALP do not differ much from the first phase ALP.

Table 5-1: The average cost in slots for the symmetric F4C2.

Largest state	4	4	4
q_f	0.2	0.3	0.4
g_1	1.86	3.22	4.96
g_{1*}	2.00	4.00	6.86
g_{2*}	2.00	3.99	6.85

The coefficients of the linear combination of the basis functions for approximating the value function are given below. There are several remarkable observations regarding the coefficients. The coefficients r_{13} and r_{24} for all (l,i) are zero. This means that the multiplication of the number of flows from different combinations does not have added value to the approximation of the value function.

Table 5-2: The weights of the linear combination of the basis functions for the symmetric F4C2, with $q_f = 0.2 \ \forall f$.

Coefficients	l=1				l=2			
	i=0	i=1	i=2	i=3	i=0	i=1	i=2	i=3
r_0	0.00	0.34	1.17	0.13	0.00	0.34	1.17	0.13
r_1	0.76	1.00	6.99	5.76	4.70	3.35	2.01	0.62
r_2	4.67	3.35	2.01	0.62	0.78	0.98	7.06	5.84
r_3	0.63	0.98	7.06	5.85	4.67	3.36	2.02	0.61
r_4	4.70	3.36	2.02	0.61	0.61	1.00	6.99	5.76
r_{11}	0.64	1.41	0.00	0.13	0.26	0.44	0.55	0.66
r_{22}	0.29	0.44	0.56	0.67	0.63	1.45	0.04	0.17
r_{33}	0.65	1.44	0.04	0.17	0.29	0.43	0.55	0.66
r_{44}	0.26	0.43	0.54	0.65	0.65	1.41	0.00	0.13

r_{12}	0.09	0.00	0.00	0.00	0.00	0.02	0.09	0.09
r_{13}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r_{14}	0.01	0.04	0.21	0.22	0.22	0.00	0.01	0.01
r_{23}	0.00	0.01	0.07	0.07	0.07	0.00	0.00	0.00
r_{24}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r_{34}	0.20	0.00	0.01	0.01	0.01	0.04	0.19	0.20

Table 5-3: The weights of the linear combination of the basis functions for the symmetric F4C2, with $q_f = 0.3 \ \forall f$.

Coefficients	**l=1**				**l=2**			
	i=0	**i=1**	**i=2**	**i=3**	**i=0**	**i=1**	**i=2**	**i=3**
r_0	0.00	0.43	1.68	0.00	0.00	0.43	1.68	0.00
r_1	0.98	1.91	7.21	6.60	5.79	4.38	2.81	0.98
r_2	5.54	4.39	2.82	0.98	0.98	1.89	6.98	6.30
r_3	0.67	1.87	6.83	6.13	5.39	4.11	2.53	0.67
R_4	5.64	4.09	2.52	0.67	0.67	1.90	7.06	6.44
r_{11}	0.82	1.28	0.00	0.00	0.13	0.40	0.60	0.82
r_{22}	0.14	0.40	0.61	0.82	0.82	1.25	0.00	0.00
r_{33}	0.83	1.22	0.00	0.00	0.14	0.42	0.62	0.83
r_{44}	0.13	0.42	0.62	0.83	0.83	1.25	0.00	0.00
r_{12}	0.41	0.02	0.06	0.07	0.08	0.09	0.32	0.38
r_{13}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r_{14}	0.06	0.07	0.26	0.30	0.32	0.01	0.05	0.06
r_{23}	0.20	0.09	0.33	0.38	0.41	0.04	0.16	0.18
r_{24}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r_{34}	0.35	0.04	0.16	0.19	0.20	0.08	0.28	0.33

Table 5-4: The weights of the linear combination of the basis functions for the symmetric F4C2, with $q_f = 0.4 \ \forall f$.

Coefficients	l=1				l=2			
	i=0	i=1	i=2	i=3	i=0	i=1	i=2	i=3
r_0	0.00	0.60	2.40	0.14	0.00	0.60	2.40	0.14
r_1	1.20	2.26	7.19	6.75	6.60	5.00	3.21	1.03
r_2	6.79	5.20	3.39	1.16	1.25	2.32	7.34	6.93
r_3	1.11	2.34	7.38	6.96	6.83	5.22	3.40	1.11
r_4	6.64	5.03	3.22	0.98	1.07	2.27	7.22	6.79
r_{11}	0.92	1.24	0.00	0.00	0.00	0.33	0.63	0.95
r_{22}	0.00	0.33	0.63	0.96	0.95	1.25	0.00	0.00
r_{33}	0.98	1.25	0.00	0.00	0.00	0.34	0.64	0.98
r_{44}	0.00	0.34	0.64	0.97	0.95	1.24	0.00	0.00
r_{12}	0.10	0.16	0.21	0.26	0.29	0.06	0.07	0.09
r_{13}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r_{14}	0.15	0.04	0.05	0.07	0.07	0.08	0.11	0.14
r_{23}	0.24	0.03	0.04	0.05	0.05	0.13	0.18	0.22
r_{24}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r_{34}	0.21	0.16	0.21	0.26	0.28	0.11	0.15	0.18

In order to evaluate the results obtained from the ALP approach, a comparison will be made with the average cost based on several control strategies obtained by simulating the chain. First, the ALP approach is compared to the FC strategy, in which the order of the served combinations is fixed and also the duration of the green periods. Further, the ALP approach is compared to the exhaustive control strategy, in which the order of served combinations is fixed and the green periods will be kept until the flows in the combination that has right of way is empty. Two alternative exhaustive controls are also added in the evaluation; XHC(1) and XHC(2), which anticipate departures during 1 and 2 yellow slots, respectively. In other words, the green periods will be kept until the number of cars at each flow in the combination that has right of way is at most one and two in XHC(1) and XHC(2), respectively.

Based on *Little's Law*, the average waiting time per car can be derived from the average number of cars waiting at the queue. Denote the average waiting time in seconds for flow f as $\mathrm{E}(W_f)$. Then,

$$\mathrm{E}(W_f) = \frac{2 \times g_f}{q_f}, \qquad \textbf{(5-8)}$$

where g_f and q_f are the average number of cars waiting at flow f and the arrival rate, respectively. The overall average waiting time is weighted by the average arrival at the queues. Thus, the overall average waiting time in seconds is given by

$$\mathrm{E}(W) = \sum_f \frac{q_f}{c} \mathrm{E}(W_f) = \sum_f 2.\frac{q_f}{c}.\frac{g_f}{q_f} = \frac{2}{c}\sum_f g_f = \frac{2g}{c}, \qquad \textbf{(5-9)}$$

where $c = \sum_f q_f$.

Table 5-5 presents the overall average waiting time in seconds for varying arrival rates at a symmetric F4C2 intersection, which means that all flows have identical arrival rates per slot of q_f.

Table 5-5: Overall average waiting time in seconds for the symmetric F4C2.

Rule	$q_f = 0.2$		$q_f = 0.3$		$q_f = 0.4$	
Two-phase ALP	5.00		6.65		8.57	
FC	5.37	7%	7.30	10%	8.92	4%
XHC	5.66	13%	7.54	13%	8.65	1%
XHC(1)	5.06	1%	6.69	1%	8.37	-2%
XHC(2)	5.16	3%	6.97	5%	8.99	5%
FC green periods in slots for C1, C2	1, 1		3, 3		8, 8	

For this simple intersection, the overall average waiting time gained from the two-phase ALP approach is less than the other strategies. The results obtained by the two-phase ALP approach is close to the anticipating exhaustive XHC(1) strategy.

5.2.2 Asymmetric arrival rates

In the case of asymmetric arrival rates, two situations are considered for F4C2. The first situation that is considered is where the arrival rates are 0.15 for flows 1 and 3, and 0.45 for flows 2 and 4. Hence, the flows within the same combination have the identical arrival rates, but C2 is three times as busy as C1. The second situation is where the arrival rate for flow 1 is 0.10, and the arrival rates for the other flows are 0.30. The coefficients of the linear combination of the basis functions obtained from the two-phase ALP approach for approximating the value function are given below.

Table 5-6: The weights of the linear combination of the basis functions for the asymmetric F4C2, with q =(0.15, 0.45, 0.15, 0.45).

Coefficients	l=1				l=2			
	i=0	i=1	i=2	i=3	i=0	i=1	i=2	i=3
r_0	0.00	0.44	0.00	0.00	0.00	1.25	2.97	0.00
r_1	2.25	1.22	9.01	8.32	7.61	5.97	4.28	2.25
r_2	6.30	6.13	4.61	0.91	0.91	3.10	6.76	6.44
r_3	2.64	1.07	9.67	8.70	8.29	6.59	4.81	2.64
r_4	6.19	5.66	4.03	0.91	0.91	3.07	6.65	6.30
r_{11}	0.62	1.76	0.00	0.00	0.00	0.19	0.39	0.62
r_{22}	0.00	0.00	0.30	0.95	0.95	0.95	0.00	0.00
r_{33}	0.68	1.97	0.00	0.00	0.00	0.21	0.43	0.68
r_{44}	0.00	0.10	0.42	0.95	0.95	0.94	0.00	0.00
r_{12}	0.08	0.01	0.01	0.01	0.01	0.03	0.06	0.07
r_{13}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r_{14}	0.03	0.00	0.00	0.00	0.00	0.01	0.02	0.03
r_{23}	0.12	0.17	0.21	0.25	0.26	0.04	0.09	0.11
r_{24}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r_{34}	0.22	0.18	0.21	0.26	0.27	0.07	0.16	0.20

Table 5-7: The weights of the linear combination of the basis functions for the asymmetric F4C2, with q =(0.1, 0.3, 0.3, 0.3).

Coefficients	l=1				l=2			
	i=0	i=1	i=2	i=3	i=0	i=1	i=2	i=3
r_0	0.00	0.31	1.05	0.00	0.00	0.47	1.44	0.03
r_1	0.55	0.00	6.22	4.98	4.18	2.87	1.71	0.55
r_2	4.93	4.04	2.46	0.68	0.68	1.81	6.33	5.52
r_3	1.42	1.66	7.82	7.23	7.08	5.57	4.02	1.41
r_4	5.26	4.02	2.43	0.69	0.69	1.88	6.96	6.36
r_{11}	0.56	1.54	0.10	0.17	0.23	0.46	0.51	0.56
r_{22}	0.10	0.36	0.59	0.79	0.79	1.13	0.00	0.00
r_{33}	0.91	1.38	0.00	0.00	0.00	0.32	0.55	0.91
r_{44}	0.16	0.34	0.58	0.78	0.78	1.23	0.00	0.00
r_{12}	0.00	0.01	0.15	0.16	0.16	0.00	0.00	0.00
r_{13}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r_{14}	0.02	0.05	0.27	0.29	0.29	0.00	0.00	0.00
r_{23}	0.66	0.21	0.24	0.28	0.30	0.14	0.51	0.61
r_{24}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r_{34}	0.22	0.24	0.15	0.18	0.19	0.05	0.17	0.20

The results are given in Table 5-8. The overall mean waiting time is denoted by $E(W)$ and the average waiting time for flow f is denoted by $E(W_f)$. The length of green periods in the FC control strategy for C1 and C2 are 1 and 5 slots, respectively, in the first situation. In the second situation, the length of the green periods is 3 slots for both combinations.

Table 5-8: Mean waiting times in seconds for two asymmetric F4C2 cases.

Rule	$E(W)$		$E(W_1)$	$E(W_2)$	$E(W_3)$	$E(W_4)$
q =(0.15, 0.45, 0.15, 0.45)						
Two-phase ALP	5.95		8.47	5.16	8.41	5.07
FC	6.32	6%	10.53	4.91	10.55	4.91
XHC	6.72	13%	9.86	5.68	9.88	5.67
XHC(1)	6.37	7%	8.12	5.79	8.11	5.79
XHC(2)	7.32	23%	7.07	7.42	7.08	7.39
q =(0.10, 0.30, 0.30, 0.30)						
Two-phase ALP	6.22		4.93	5.43	8.25	5.40
FC	7.09	14%	5.19	7.30	7.31	7.30
XHC	6.96	12%	6.42	6.66	7.72	6.67
XHC(1)	6.22	0%	5.25	6.05	6.88	6.06
XHC(2)	6.61	6%	4.63	6.65	7.19	6.65

As observed, the two-phase ALP approach results a lower overall mean waiting time compared to the other strategies. Further, it is observed that the flows that are parts of the more busy combination have a lower waiting time. This gives an idea that a more busy combination gets the priority. If a busy flow and a less busy flow are grouped together in one combination, then the less busy flow will take advantage of the priority and the more busy flow will suffer a bit.

5.3 Reduced Linear Program

Although the dimension of the problem is reduced in the approximate version (ALP), the number of constraints is still as many as the number of state-action pairs which can be very large. The two-phase ALP only solves parts of the '*curse of dimensionality*' problem. Therefore, the largest state on each flow is limited to 4. In practice, this is certainly not the case. Van Roy **[2]** suggested a constraint sampling approach for approximating solutions to optimization problems when the number of constraints is intractable, the *Reduced Linear Program* (RLP). The idea is to define a probability distribution Φ over the set of constraints and to include only a subset of the distributed constraints for solving the problem. Two properties were proven that if a reasonable number of constraints are sampled from distributed constraints, then almost all others will be satisfied and the constraints that are not satisfied do not distort the solution too much.

The RLP for the traffic control problem is characterized as follows. The largest state on each flow is set to 10. The constraint sample size is 20,000. The subset of constraints were sampled based on probability measure $\Phi(\underline{k}) = (1-\varrho)^4 \varrho^{|\underline{k}|}$, $\varrho = 0.99$. In the case of small arrival rates, say 0.2, the RLP gives a good approximate to the value function that results the average cost similar to average cost yielded from the two-phase ALP. However, when the load is high, the results do not resemble the expectations because the

sampled constraints do not represent "almost all" other constraints, and the constraints that are not satisfied distort the solution effectively. It is proven that the results of RLP rely on an idealized choice of Φ. The underlying thought is similar as choosing the state relevance weights (see Section 3.3). The constraints that represent the states that are visited more often should be then included. The chosen probability measure $\Phi(\underline{k})$ has an exponential form which gives high probabilities to the states that are close to the situation when there are no cars present in all flows, i.e., the states with small number of cars present in the flows. This probability measure is an acceptable choice of Φ when the arrival rates are small. However, when the arrival rates are higher, the high probability has to be assigned to the states where more cars are present in the flows. Therefore, exponential form is not an idealized choice of the probability measure Φ.

6 Conclusion

The two-phase ALP approach for average cost dynamic programming is applied to the traffic control at isolated signalized intersection. As an illustration, an intersection of two two-way streets with controllable traffic lights on each corner is considered. Based on the chosen basis functions and the state relevance weights, it is observed that the results based on the switching schemes obtained by the ALP approach were better compared to the other strategies where there are at most 4 cars on each flow.

In order to deal with the intractable state space, the ALP approach provides an algorithm for finding a good approximation to the optimal value function by fitting a parameterized linear function. Hence, the number of variables that has to be stored in ALP approach is equal to the number of the pre-specified basis functions, instead of storing the optimal value function itself for each state in the system. However, the ALP approach still has as many constraints as the exact linear programming formulation. Therefore, the largest state on each flow is limited to 4. In practice, this is certainly not the case. Van Roy **[2]** suggested a constraint sampling approach for approximating solutions to optimization problems when the number of constraints is intractable. The idea is to define a probability distribution Φ over the set of constraints and to include only a subset of the distributed constraints for solving the problem.

References

[1] Farias, D.P. de, Benjamin van Roy, "Approximating Linear Programming for Average-Cost Dynamic Programming".

[2] Farias, D.P. de, Benjamin van Roy, "On Constraint sampling in the Linear Programming Approach to Approximate Dynamic Programming". *Mathematics of operations research, Vol. 29, No. 3, August 2004, pp. 462-278*.

[3] Farias, D.P. de, (June 2002), "The Linear Programming Approach to Approximate Dynamic Programming: theory and application".

[4] Haijema, R, Jan van der Wal (3rd November 2006), "An MDP decomposition approach for traffic control at isolated signalized intersections".

[5] Koole, G (9th September 2005), "Lecture notes Stochastic Optimization", Lecture notes VU Amsterdam.

[6] http://en.wikipedia.org/wiki/Traffic_light#Introduction

[7] "Intersection Safety: Myth Versus Reality", Intersection Safety Brief, U.S. Department of Transportation, Federal Highway Administration.