Qoordination: Learning to Coordinate Traffic Controller Agents in Dynamic Transport Networks

Derek Fagan

A thesis submitted to the University of Dublin, Trinity College
in fulfillment of the requirements for the degree of
Doctor of Philosophy (Computer Science)

April 2015

Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise stated, it is entirely my own work. I agree that Trinity College Library may lend or copy this thesis upon request.

Derek Fagan

Dated: April, 2015

Acknowledgements

"Knowledge without labor is profitless.
Knowledge with labor is genius."
- Gordon B. Hinckley

As I sit here, my wife and I watch our children run and play together, enjoying some well-deserved Rest and Relaxation time (R&R)*. Upon reflection my mind is drawn back to my PhD research that is at length coming to an end, as well as to the people and organizations that made this research possible.

My children have all been born during my years at college. I was already in my third year of college when my first son, Brighton, was born. Evian arrived one year later, and then during my masters year Livvy arrived. I must admit that when I got the phone call to quickly make my way to the hospital for Livvy's birth I was in the middle of class on the verge of standing up to give a presentation that I was extremely unprepared for. Liena, Lehi, and Lidia were all born during the years that I spent working on my PhD. I would like to thank all of my dear, sweet children for bringing me so much joy and adventure over these past few years of study.

I would like to thank my parents for their unending supply of love and support throughout my studies. I feel that you have blessed me with the privilege of a full education, both academically and in life in general.

I would like to thank my sister in law, Geraldine, for giving me the idea to start this PhD in the first place.

I would like to thank all of the organizations that financially provided for me and my family throughout the course of my PhD. To Conor and Eamonn of uTrack Ltd. I know that supporting an employee with such large family commitments couldn't have been easy for a startup company. Thank you. To the Irish Research Council for the scholarship funding provided. To HSLU for the flexibility provided me to be able to finish this PhD.

I would like to thank my supervisor, Dr. René Meier, for his expertise and guidance. Thank you for all of the time and effort that you have put into supervising my PhD.

I would like to thank Dr. Donal O'Mahony who stepped in when I needed him and really helped to get the thesis through its finishing stages.

I would like to thank all of those who reviewed my papers and who read through and gave feedback on this thesis.

I would most especially like to thank my wife Eve, who has been beside me the whole time. Always supportive of my decisions and never faltering in her encouragement. Thank you for being there.

I would not be being true to myself if I did not acknowledge the help of my Lord and my God. I don't believe that I would have been able to make it without heaven's help.

The children are still playing so I'm going to go now and spend some time with them while the day lasts.

* Having spent so long writing this thesis I find it difficult to stop abbreviating Common Terms (CTs)*

Derek Fagan
University of Dublin, Trinity College
April 2015

Abstract

The ultimate objective of traffic control is to optimize the flow of traffic throughout the transport network. Optimized traffic flow leads to reduced traffic congestion and consequently reduced travel times, accidents, noise and air pollution. Approaches to traffic control must be adaptive so as to be able to learn how to optimize for traffic flow levels which range from light to heavy or that change significantly from one minute to the next. These approaches must also be able to optimize traffic flow within transport networks that range in size from a single individual intersection to networks containing hundreds of connected intersections. Coordination of traffic signal controllers along busy traffic corridors enables the establishment of progressive signal systems. These systems create "green waves" of traffic such that vehicles traveling along the main traffic corridors of the network are unlikely to have to stop at any red lights. Establishing progressive signal systems is not a trivial task. In order to do so the traffic light sequences of intersections along a traffic corridor must have their start time offset from each other. The appropriate offset is traditionally calculated based upon the distances between the intersections and the average speed of vehicles traveling between the intersections. It is the time consuming and error prone job of a traffic engineer to manually specify these variables for each individual intersection. Further complications are introduced by the fact that traffic flow within a transport network is dynamic by nature. For example the reversal of traffic flow direction between morning traffic peak and evening traffic peak would require a complete reversal of the progressive signal system. Such traffic flow patterns may change over time as the urban environment continues to develop. Another example arises in the event of road closure, which could lead to short-term diversions in the path of a traffic corridor. Whereas long-term alterations can be dealt with by regular maintenance visits from a traffic engineer it is often not possible to foresee and handle short-term changes.

We theorize that intersection agents within multi-agent transport networks can automatically create dynamic progressive signal systems using Reinforcement Learning (RL) techniques.

In this thesis we present a learning based approach to signal coordination that we refer to as Qoordination. This approach uses an RL technique named Q-Learning. Qoordination automates the establishment and maintenance of dynamic progressive signal systems. It does this by firstly detecting

key intersections within the network. Key intersections are those through which relatively high traffic flows enter the network. All non-key intersections base some of their signal timing changes on the signal timings of adjacent upstream neighbor intersections so as to maintain coordination. The offsets of all non-key intersections are adjusted so as to minimize their local vehicle queue lengths. As queue lengths are minimized throughout the network progressive signal systems begin to automatically emerge along the main traffic corridors. The paths and directions of these progressive signal systems adjust themselves automatically as traffic flow changes. Each intersection can then use any one of a number of different traffic control methods to optimize their own local throughput. Such traffic control methods include Round Robin, SAT, or even other learning based methods.

In this thesis we also introduce a Multi-Layer Hashing (MLH) method of function approximation that is ideal for use in RL algorithms. This technique enables rapid learning in large dynamic multi-agent environments such as the transport networks addressed in this thesis. This rapid learning is achieved through generalization and abstraction in a way that is intuitive and straightforward to implement and that allows for the extension of RL algorithms without significantly altering them.

Qoordination is evaluated using an evaluation platform into which the industry standard VISSIM microscopic simulator is integrated. Evaluation simulations are performed with both static and dynamic traffic loads. These simulations are also performed with both static and dynamic traffic corridor paths. We evaluate the scalability of different Qoordinated and non-Qoordinated approaches to traffic control in a novel fashion by executing them within simulated transport networks of incrementally increasing size.

Our results show that for a variety of traffic control methods Qoordination leads to significant improvements in network traffic flow in terms of vehicle queue lengths and average vehicle waiting times. The level of improvement seen is proportional to the network size, the traffic flow level, and the complexity of the main traffic corridors within the transport network. In our evaluation experiments Qoordination leads to reductions in average vehicle waiting times of up to 61% under light traffic flow levels when compared to uncoordinated control. This figure is increased to 76% under heavy traffic flow. This figure is further increased to 83% with increased complexity in the path of the main traffic corridor.

Contents

Acknowledgements **iii**

Abstract **v**

Contents **vii**

Chapter 1 Introduction **1**

1.1 Traffic Control 1

1.2 Reinforcement Learning 3

1.3 Research Questions 4

1.4 Principal Contributions 5

1.5 Thesis Overview 6

Chapter 2 Background and Related Research **7**

2.1 Intelligent Rational Agents 7

2.2 Agent Based Learning 11

2.2.1 Markov Decision Processes 11

2.2.2 Learning Algorithms 14

2.2.2.1 Dynamic Programming 14

2.2.2.2 Adaptive Dynamic Programming 16

2.2.2.3 Temporal Difference Learning 17

2.2.3 Policy Definition 21

2.2.3.1 Policy Representation 21

2.2.3.2 Multiple Policies 25

2.2.4 Representation 27

2.2.4.1 Lookup Tables 27

2.2.4.2 Function Approximation 28

2.2.4.3 Conclusion 37

2.2.5 Exploration 40

2.2.5.1 Greedy *40*
2.2.5.2 ϵ-Greedy *41*
2.2.5.3 Boltzmann *42*
2.2.6 Coordination Graphs *42*
2.3 Traffic Control 44
2.3.1 Traffic Engineering Concepts and Terminology *45*
2.3.2 Classical Approaches to Traffic Control *51*
2.3.2.1 Pre-timed *51*
2.3.2.2 Actuated *51*
2.3.2.3 Adaptive *52*
2.3.3 Artificial Intelligence Based Traffic Control *54*
2.3.3.1 Centralized *54*
2.3.3.2 Independent Agents *55*
2.3.3.3 Coordinated Agents *56*
2.3.3.4 Reinforcement Learning Based Approaches *57*
2.3.4 Conclusion *60*
2.4 Summary 65

Chapter 3 Qoordination **66**
3.1 Requirements 66
3.2 Motivations 67
3.2.1 Architecture *67*
3.2.2 Technology *68*
3.2.3 Environment *69*
3.2.4 Learning Algorithm *71*
3.2.5 Policy Definition *72*
3.2.6 Representation *74*
3.2.6.1 Particle Filtering *74*
3.2.6.2 Neuro-Fuzzy Network *75*
3.2.6.3 Artificial Neural Network *75*
3.2.6.4 High Dimensional Clustering *76*
3.2.6.5 Multi-Layer Hashing *76*
3.2.6.6 Summary *76*
3.2.7 Reward Scope *77*
3.2.8 Irrelevant Agent Abstraction *77*
3.2.9 Exploration Scheme *78*
3.2.10 State *79*
3.2.11 Action *80*
3.2.12 Reward *80*
3.2.12.1 Throughput *80*

3.2.12.2 Vehicle Speed ... *81*
3.2.12.3 Vehicle Waiting Time ... *81*
3.2.12.4 Queue Length ... *81*
3.3 Multi-Layer Hashing ... 82
3.3.1 Hashing ... *82*
3.3.1.1 State Variable Hashing ... *82*
3.3.1.2 State Hashing ... *82*
3.3.2 Layers of Granularity ... *83*
3.3.3 Inner Action Hash Tables ... *84*
3.3.4 Exploration ... *85*
3.3.5 Error ... *85*
3.3.6 Action Selection ... *86*
3.3.7 Update Algorithm ... *87*
3.3.8 Layer Abstraction ... *89*
3.3.9 Layer Parameter Space Searching ... *89*
3.3.10 Conclusion ... *90*
3.4 Process of Qoordination ... 91
3.4.1 Detect Key Intersection ... *92*
3.4.2 Phase Length Modifications ... *94*
3.4.3 Offset Modifications ... *94*
3.4.4 Agent Abstraction ... *95*
3.5 Summary ... 96

Chapter 4 Implementation and Simulation-Based Evaluation Platform 97

4.1 Evaluation Platform ... 97
4.1.1 VISSIM Microscopic Simulator ... *97*
4.1.2 Network Generation ... *99*
4.1.2.1 Single Intersection ... *99*
4.1.2.2 Multiple Intersections ... *101*
4.1.2.3 Parameter File ... *102*
4.1.2.4 Incremental Network Size Increase ... *103*
4.1.3 Report Generation ... *103*
4.1.4 Automated Evaluation ... *104*
4.2 Traffic Control Framework ... 105
4.2.1 Integration with Evaluation Platform ... *106*
4.2.2 Signal Controller ... *109*
4.2.2.1 Initialization ... *109*
4.2.2.2 Update ... *110*
4.2.2.3 Action Performance ... *110*
4.2.3 Agent ... *111*

4.2.3.1 State and Actions ... *111*
4.2.3.2 Abstract Classes ... *111*
4.2.3.3 Initialization ... *112*
4.2.3.4 Update ... *112*
4.3 Qoordination Agents ... 113
4.3.1 Initialization ... *114*
4.3.2 Get Value ... *115*
4.3.3 Update Values ... *116*
4.3.4 Action Selection ... *116*
4.4 Conclusion ... 116

Chapter 5 Evaluation and Analysis **118**
5.1 Objectives ... 118
5.2 Metrics ... 119
5.3 Evaluation Transport Networks ... 120
5.4 Traffic Control Methods ... 121
5.4.1 Round Robin ... *122*
5.4.2 SAT ... *123*
5.4.3 Q-Learning Based Traffic Control ... *124*
5.5 Parameters ... 124
5.6 Experiments ... 125
5.6.1 Experiment 1 – Increasing Network Size and Static Traffic Flow Levels ... *125*
5.6.1.1 Comparison of Traffic Control Methods in a Single Intersection ... *126*
5.6.1.2 Effects of Network Size on Uncoordinated Traffic Control Methods ... *135*
5.6.1.3 Pre-Coordinated Good / Bad ... *143*
5.6.1.4 Actuated Traffic Control ... *143*
5.6.1.5 Effects of Network Size and Coordination on Traffic Control Methods ... *145*
5.6.1.6 Conclusion ... *152*
5.6.2 Round Robin Justification ... *154*
5.6.3 Experiment 2 – Changing Traffic Flow Direction ... *154*
5.6.3.1 Results and Analysis ... *154*
5.6.4 Experiment 3 – Changing Traffic Flow Levels ... *156*
5.6.4.1 Results and Analysis ... *156*
5.6.5 Experiment 4 – Reversing a Single Progressive Signal System ... *158*
5.6.5.1 Results and Analysis ... *159*
5.6.6 Experiment 5 – Complex Progressive Signal System ... *161*
5.6.6.1 Results and Analysis ... *161*
5.6.7 Conclusion ... *163*
5.7 Multi-Layer Hashing Utility Function Analysis ... 164
5.7.1 Initialization Process ... *166*

5.7.2 Agent 3 ..... *166*
5.7.2.1 After Initialization *167*
5.7.2.2 After Extended Use *168*
5.7.2.3 Optimized *169*
5.7.3 Agent 1 ..... *169*
5.7.3.1 After Initialization *170*
5.7.3.2 After Extended Use *170*
5.7.3.3 Optimized *171*
5.7.4 Agent 2 ..... *172*
5.7.5 Agent 0 ..... *173*
5.7.6 Conclusion *174*
5.8 Summary 174

Chapter 6 Conclusion and Future Research **176**
6.1 Summary 176
6.2 Contributions 177
6.3 Future Research 179

Bibliography **181**

Chapter 1

Introduction

This chapter introduces the domain of traffic control and gives a brief description of Reinforcement Learning (RL). The research questions that this thesis addresses are then introduced, as are the principal contribution of this work. This chapter then concludes by giving an overview for the remainder of this thesis.

1.1 Traffic Control

With constantly increasing numbers of road users traffic congestion is a significant problem that is steadily worsening. Negative effects of traffic congestion include increased commuter travel times, increased fuel consumption, increased risk of road accidents, and increased noise and air pollution levels.

In a study carried out within 437 urban areas within the USA it was found that congestion costs increased from $14.9 billion to $78.2 billion between 1982 and 2005 (Schrank & Lomax, 2007). Within this time these same areas also saw travel delay increase from 0.8 billion hours to 4.2 billion hours as well as "wasted" fuel increase from 0.5 billion gallons to 2.9 billion gallons. By 2005 it was estimated that traffic signal coordination alone had saved $451 million in congestion costs and had reduced delay by 21 million hours within these urban areas. In this same study it was also estimated that if signal coordination were implemented on all roads within the studied area then commuter travel delay could have been reduced by a further 34.5 million hours.

A variety of traffic control methods are available that range in their effectiveness and their complexity. These methods also range in their ability to adapt to changing traffic flow levels and their

ability to establish coordination among intersections. The most common method of traffic control is called pre-timed control (Roess, Prassas, & McShane, 2011). Traffic controllers that implement pre-timed control change their traffic light timings based upon the time of day and day of week. These timing plan schedules are configured by traffic engineers, typically by entering historical traffic flow data for the intersection into traffic flow optimization software such as TRANSYT (Robertson, 1969). Timing plans can also be made for multiple intersections that are to operate in coordination with each other. In these situations the offset variables are also optimized. This can result in progressive signal system establishment within main traffic corridors whose characteristics are regular i.e. they form at the same time each day, following the same path, and moving in the same direction. Pre-timed traffic control cannot however dynamically adapt to unanticipated changes in either traffic flow levels or traffic corridor course or direction. In this thesis we refer to uncoordinated pre-timed traffic control as the Round Robin method. One of the major benefits of pre-timed traffic control is its cost effectiveness and reliability due to its lack of reliance on sensors.

Another common method of traffic control is that of actuated control (Roess, Prassas, & McShane, 2011). Traffic controllers that implement actuated control change their traffic light timings dynamically based upon real-time actuator readings. Sensors are typically embedded in the road just upstream of the intersection. Actuation based intersection controllers typically cannot coordinate their actions to establish progressive signal systems along main traffic corridors due to the dynamic nature of their traffic light timing plans. For this reason actuated traffic control is typically used at isolated intersections.

Methods that have higher computing requirements than pre-timed and actuated control are referred to as adaptive traffic control methods (Roess, Prassas, & McShane, 2011). A number of different methods have been suggested for adaptive traffic control. Early methods required traffic engineers to specify the signal timing plans to be applied in a variety of different situations (Sims & Dobinson, 1980). These timing plans are then activated based upon sensor readings as opposed to time of day. The process of generating accurate timing plans is both error-prone and costly. Unforeseen circumstances such as local music events or road closure cannot typically be adapted to using these methods. Later adaptive methods apply Artificial Intelligence (AI) techniques to traffic control. These traffic control systems range in architecture from centralized (Tomforde et al., 2008) to hierarchical (Srinivasan, Choy, & Cheu, 2006) to fully decentralized (Xie, 2007). Although decentralizing the architecture alleviates issues of scalability and single points of failure it also makes it more difficult to achieve coordination between multiple intersections. Agent-based architecture is a decentralized AI approach where each intersection controller is modeled as an autonomous intelligent agent (Bazzan, 2009). RL is an agent-based technique that has shown particular promise in the area of traffic flow optimization. This is due to the fact that it does not require a predefined model of the traffic network environment. RL also has the benefit of being able to take into account both short-term and long-term rewards of intersection agent actions. In its original

single-agent form RL can be used to effectively optimize traffic flow through a single intersection (Thorpe, 1997). Coordination of intersection actions is not typically taken into account in such systems. Multi-agent techniques can be used to take coordination into account when optimizing traffic flow through multiple intersections (Roess, Prassas, & McShane, 2011) (Salkham, Cunningham, Garg, & Cahill, 2008). An area of research that has not of yet received attention is the use of RL techniques in establishing the coordination of multiple intersection agents that each implements one of a variety of different methods of traffic control. Such is the topic of this thesis.

1.2 Reinforcement Learning

RL is an AI technique that, analogous to natural intelligence, learns over time through trial and error interaction with the surrounding environment (Watkins & Dayan, 1992) (Sutton & Barto, 1998) (Russell & Norvig, 2010). RL is an unsupervised learning technique and as such, does not require the tutelage of an instructing domain expert. Instead, an RL agent autonomously explores its surrounding environment. The RL agent firstly senses the current state of the environment. It then chooses and executes an action or a sequence of actions that are available to it. It then receives a numerical feedback called a reward from the environment for the actions that it has performed. Over time the RL agent uses these rewards to build up a utility function. This process of an RL agent learning through interaction with its environment is illustrated in Figure 1.

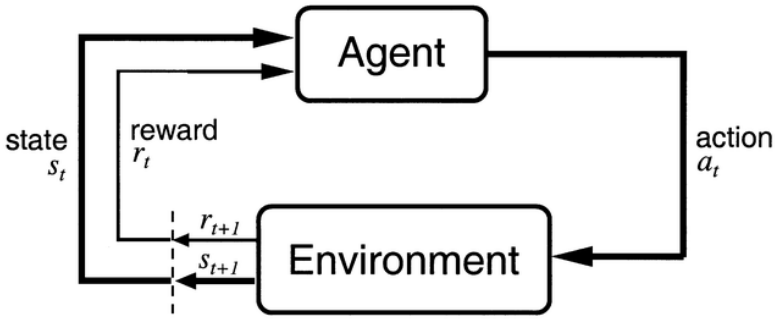

Figure 1. Process of RL agent learning through interaction with its environment (Sutton & Barto, 1998)

A utility is a numerical representation of the sequence of rewards that the agent can expect to receive when executing a series of actions starting with a given action being taken from a given state. An action's utility can thus be seen as its long term reward. The RL function known as the policy is used to decide which action is to be taken from the current state. Explorative actions can be chosen to be executed so as to increase the accuracy of the agent's utility function. Exploitative actions can also be selected for execution. These actions exploit the utility function's knowledge and are typically actions that lead to the highest long term reward when executed from the agent's current state.

In its original form RL is a single-agent algorithm. A single RL agent operates on a local level, gaining an understanding of the local effects of its own local actions. Multi-agent algorithms however also exist that enable multiple agents to work together to yield higher global utility values (Shoham & Leyton-Brown, 2008) (Vlassis, 2007) (Claus & Boutilier, 1998) (Weiss, 2013).

1.3 Research Questions

Research Question RQ1: How can autonomous intersection agents learn to coordinate their actions so as to create dynamic progressive signal systems within dynamic transport networks? Automated coordination of traffic control agents within a transport network is a non-trivial task (Guestrin, Lagoudakis, & Parr, 2002). Automatic formation and maintenance of progressive signal systems is particularly challenging due to the dynamic nature of traffic flow. This dynamic nature is made manifest as traffic flow levels change significantly not only from hour to hour but also from month to month and from year to year. It is also made manifest by fluctuation in both the direction and the course of many main traffic corridors running through transport networks. Course changes in main traffic corridors can be short term, such as occur due to road closures and diversions. They can also be long term, as can occur due to expansion of the transport infrastructure. We theorize that intersection agents within multi-agent transport networks can automatically create dynamic progressive signal systems using Reinforcement Learning (RL) techniques. Further, we theorize that not only does coordination of traffic control lead to improved transport network performance (with regard to vehicle waiting times and queue lengths) but that the improvement in performance is directly related to the network's size. Thus the larger the network is the greater the benefit of coordinating traffic control within it. We expect that the establishment of coordinated traffic control within a transport network will lead to traffic congestion being kept to the outside of the coordinated area while within the coordinated area traffic will be much more free flowing.

Research Question RQ2: How can RL agents learn accurate utility functions within dynamic multi-agent environments? Learning within a multi-agent transport network is a non-trivial task. This is particularly due to the dynamic nature of such an environment and to the interdependencies that exist between the highly coupled adjacent intersection agents. An appropriate RL utility function would need to be able to cater for the non-static nature of traffic flow levels. Due to the interdependencies of intersection agents certain state variables need also be used in neighboring agent state spaces. This can result in relatively large state spaces for agents with many neighboring intersections. This increase in state space size can result in overly long training times. An appropriate utility function should thus be able to generalize previously

acquired information to unseen circumstances. Efficient generalization reduces agent learning times (Ponsen, Taylor, & Tuyls, 2010). Intersection agents are not always dependent on all of their adjacent neighbor intersections. A utility function that can take this into account through abstraction can further reduce state space size and in turn again reduce learning times (Ponsen, Taylor, & Tuyls, 2010). More advanced utility function representations, which are commonly referred to as function approximators often add significant complexity and in turn typically necessitate major modifications to RL algorithms (Sutton & Barto, 1998) (Dayan, 1992) (Tsitsiklis & Van Roy, 1997). In this thesis we seek to discover an appropriate function approximation method that can efficiently perform generalization and abstraction on stochastic input in a way that can be easily adapted to RL algorithms.

1.4 Principal Contributions

Contribution C1: Qoordination. Qoordination is a novel Q-Learning based approach to coordinating traffic control agent actions. Qoordination enables the creation and maintenance of progressive signal systems along main traffic corridors that run through transport networks. Due to the dynamic nature of traffic flow both the direction and course of these traffic corridors can change over time, as has been identified in Research Question RQ1. Qoordination works by firstly detecting intersections through which relatively high traffic flows enter the network. These are referred to as key intersections. All non-key intersections base some of their signal timing changes, namely their cycle length, which will be discussed in the next chapter, on the signal timings of adjacent upstream neighbor intersections so as to maintain coordination. All non-key intersections then adjust their offsets so as to minimize their own local vehicle queue lengths. As queue lengths are minimized throughout the network progressive signal systems begin to automatically emerge along the main traffic corridors. The direction and course of each progressive signal system adjusts itself automatically as traffic flow through the corridor changes. Qoordination can establish progressive signal systems across arterials of intersections that each employs one of a number of different traffic control methods. These traffic control methods employ different techniques to optimize their intersections own local throughput. Such traffic control methods include Round Robin, SAT, or even other learning based methods.

- **Contribution C2: Multi-Layer Hashing (MLH) function approximation method**. MLH is a novel locality-sensitive hashing inspired technique for rapid learning. It has been designed as a method of function approximation that is ideal for use in RL algorithms. MLH enables rapid learning in large dynamic multi-agent environments such as the transport networks addressed in this thesis. MLH's overlapping layers enable generalization such that expected utility values can

be predicted from states that have not yet been visited. Over time these generalizations are refined and replaced with utility values of performing the different actions from increasingly specific states. This is done in a way that is intuitive and straightforward to implement and that allows for the extension of RL algorithms without significantly altering them. MLH allows for further increases in learning times by abstracting away irrelevant information during the learning process. For example, if the state of intersection A's downstream neighbor, intersection B, is detected to have little or no bearing on A's reward then it can be weighted in A's MLH utility function to reflect this. This removes a dimension from A's MLH utility function and thus increases A's learning rate.

- **Contribution C3: Solid Evaluation Method**. Qoordination is evaluated using a novel traffic control evaluation platform. The industry standard VISSIM microscopic simulator (Fellendorf & Vortisch, 2010) is fully integrated into this novel platform so as to ensure accurate and reliable results. This platform evaluates the effects of coordination on various traffic control approaches in a novel fashion. This is done by automatically generating numerous simulated transport networks of incrementally increasing size. Various methods of traffic control are then run within these networks both with and without Qoordination. These methods of traffic control are as follows: Round Robin (Salkham & Cahill, 2010), a basic SCATS (Sims & Dobinson, 1980) based adaptive approach to traffic control named SAT (Richter, 2006), and a single-agent Q-Learning based traffic control method. Simulations are performed under a variety of traffic loads as well as in situations in which the direction of the main traffic corridors are changed. The evaluation platform manages the non-trivial task of fully automating the entire evaluation process from generating the simulated transport networks, to running the simulations under various traffic loads, to organizing and compiling the evaluation results for all traffic control approaches.

1.5 Thesis Overview

The remainder of this thesis is organized as follows. Chapter 2 presents theoretic background information and related research in the field of RL. This chapter also introduces the domain of traffic engineering and reviews existing approaches to traffic control. Chapter 3 describes in detail our Qoordination approach to traffic controller coordination. Special emphasis is given to Qoordination's novel MLH utility function. Chapter 4 describes our simulation-based evaluation platform and our Qoordination agent implementation. A description of our evaluation experiments and an analysis of the results obtained are given in chapter 5. Chapter 6 presents the conclusions of this thesis and discusses possible future work.

Chapter 2

Background and Related Research

In this chapter we introduce the background information necessary for understanding our approach to learning based traffic coordination. We also discuss related research that will help to put our approach into perspective. We describe in detail agent based learning techniques with particular emphasis on Reinforcement Learning (RL) techniques. We introduce concepts in traffic control and explain the meaning of relevant traffic engineering terminology. We then review different classical approaches to traffic control. This is then followed by a review of a variety of Artificial Intelligence (AI) based traffic control systems, with a focus on RL agent based systems.

2.1 Intelligent Rational Agents

An agent is any entity that can perceive its environment using sensors and that can act upon that environment using actuators. This concept is illustrated in Figure 2.

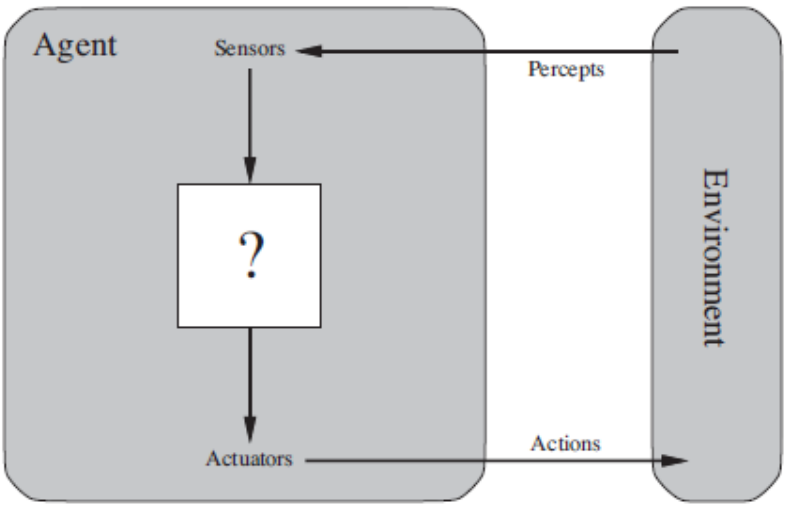

Figure 2. Agent interacting with environment (Russell & Norvig, 2010)

For example, a human agent perceives the surrounding environment using biological sensors such as eyes and ears and then acts upon that environment using biological actuators such as hands and feet. A robotic agent uses mechanical sensors such as a camera or an audio input device to perceive its environment. It can then act upon that environment using mechanical actuators such as wheels or mechanical arms. Agents need not be tangible as were the previous two examples but could also simply consist of software components. These agents can perceive their environment through received input such as network packets or user input and can then act upon their environment by sending network packets or messages to be displayed to the user. Within the area of traffic control there can be a number of different types of agents. The main two types of agents that are usually modeled in a traffic control system are vehicular agents and traffic light controller agents. A vehicular agent perceives its environment using sensors such as GPS receiver, infra-red receiver, proximity detector, camera, etc. It can then act upon its environment by sending messages to the driver, by sending messages over a connected network, or even by directly controlling the vehicle itself, depending on the level of automation of the actual vehicle. Traffic light controller agents perceive the environment using embedded sensors such as induction loops, radar detectors, or video cameras. These agents then act on their environment by changing the sequencing and timing of the traffic light signals. In this thesis only traffic light controller agents are modeled and it is assumed that all vehicles are under the complete and independent control of their human driver.

Each agent must process the perceptual inputs (percepts) received from its sensors, in which process it derives the current state of the environment. Note that the state that is derived from the percepts is that of the environment and not of the agent. The agent then receives a performance measure (reward) from the environment that evaluates the transition in environment state between the previous and current time steps. How this reward is calculated is a crucial factor to a successful agent design. The environment state

transition occurs as a result of (or at least partially as a result of) the action that the agent has most recently performed. The agent can thus learn to estimate the reward that it can expect to receive for its performance of any action from any given state. Since the derived environment state is represented and not an arbitrary agent state the learned mapping ensures that the agent does not delude itself into believing that it is doing well when it actually isn't. An agent is said to be rational if it selects actions that it expects will maximize its reward, given the current environment state and its understanding of how its environment works. This can even include random action selection in situations where the agent believes that this will lead to a higher reward. In situations where an agent is unable to calculate an expected reward for a given state and action pair then it is obliged to explore its options through experimentation.

Rational agent structures range in complexity from simple reflex agents, to much more complex goal and utility based agents. Reflex agents make rapid action selection decisions based solely on the current state of the environment. Although the rapidity of action selection for these agents is of great benefit they can be quite limited when it comes to learning in stochastic and partially observable environments (which will be explained shortly) due to excessive state and action space sizes. Goal based agents base their action selection decisions on not only the current state of the environment but also upon the goal that they are currently trying to accomplish. The agent thus maintains information regarding what it is actually trying to accomplish and can then learn the paths to achieving this goal. A utility based agent can further distinguish how good each of these paths are with respect to the performance measure i.e. the path's 'utility', and can then choose which path is best. Figure 3 illustrates the workings of a utility based agent.

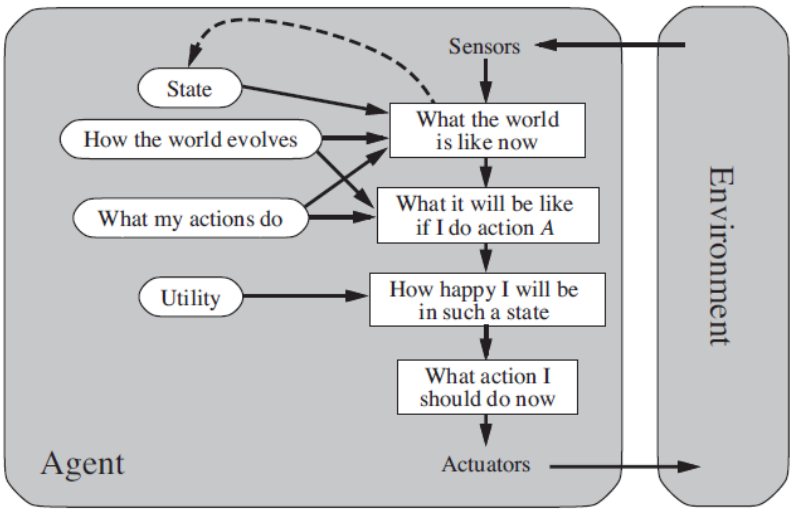

Figure 3. Utility based agent design (Russell & Norvig, 2010)

One of the single most important factors to be taken into consideration when designing an intelligent agent is the definition of the environment in which the agent will reside. There are a number of important properties to be taken into account when specifying this definition. We will now outline a list of these

properties as identified by Russell and Norvig in their seminal book "Artificial Intelligence, a modern approach" (Russell & Norvig, 2010, p40):

Single agent vs. multi-agent. In a single agent system the agent that inhabits the environment considers that all changes to the environment state over time come as a result of either that agent's own actions, the workings of the environment, or random variances. An agent within a multi-agent system however must also take into account both how the environment state is affected by the actions of other agents, and how their perceptions of the environment state change as a result of its actions. This matter is further complicated by the fact that these agents are all learning simultaneously, and so their behaviors change in response to the changing behaviors of the other neighbors in the environment. The environment thus becomes quite a dynamic abode in which to learn.

Fully observable vs. partially observable. An environment is fully observable if the agent's sensors can detect all aspects of the environment that are relevant for the agent to choose the appropriate action to take. Thus the agent is fully informed of the environment state. Noisy or missing sensor data leads to partial observability of an environment.

Deterministic vs. stochastic. A deterministic environment is one in which there is no random variance in transitioning from one state to another. Thus the next environment state can be completely determined by the current one. Stochastic environments on the other hand have an element of randomness to their state transitions. Partially observable environments and those that are inhabited by multiple agents often appear to be stochastic.

Epsodic vs. sequential. In episodic environments the transition from one state to another is not dependent on the action taken by the agent. The agent is thus presented with a series of atomic situations, or episodes, in which it is presented with precepts, selects an action, and then receives a reward for the action performed. The next episode to be presented to that agent is not dependent on the chosen action. Sequential environments however present an agent with sequences of situations wherein the action decisions of the agent will have a direct effect on the next situation presented to it. Thus the agent's sequential actions directly influence the environment state transitions.

Discrete vs. continuous. An environment's state space is said to be discrete if there is a finite number of possible states. Likewise, the agent's action space is said to be discrete if there is a finite number of actions for the agent to choose from. Continuous environments have an infinite range of states or actions and often use decimal representation for such. Whereas a chess playing agent lives within a discrete environment one that controls a robot (for which the angles at which its actuators can be placed or the force with which its actuators can be applied are continuous) would reside in a continuous environment.

Static vs. dynamic. A static environment is one which does not change over time without the interaction of the agent. Dynamic environments on the other hand will change one way or another with or without an agent performing an action.

Known vs unknown. In a known environment the agent already has an understanding of what the outcomes of its actions are going to be. In an unknown environment the agent must explore through experimentation to gain such knowledge.

The approach taken in this thesis is to model the environment as a multi-agent, fully observable, stochastic, sequential, discrete, dynamic, unknown environment (see section 3.2.3).

2.2 Agent Based Learning

In this section we will discuss concepts and techniques essential to agent based learning.

2.2.1 Markov Decision Processes

The challenge of learning to optimize an agent's utility over the course of a set of sequential decisions can be efficiently addressed by modelling the problem as a Markov Decision Process (MDP) (Howard, 1960). An MDP can be defined as a discrete time stochastic control process, which provides a mathematical framework for decision-making. An MDP is characterized by the following:

- *S*: State space, which consists of a set of states that the process may be in
- *A(s)*: Action space, which consists of a set of possible actions that can be taken from state *s*
- *P(s'|s,a)*: Transition function, which gives the probability that performing action *a* will lead the process to transition from state *s* to state *s'*
- *R(s)*: Reward function, which returns the short term reward from the environment for being in state *s*

Figure 4 gives an example of a simple MDP that has three states (s) and three actions (a). Each arrow leading from an action to a resulting state is labeled by the transition probability (P). The reward for being in any given state S is represented by the letter R.

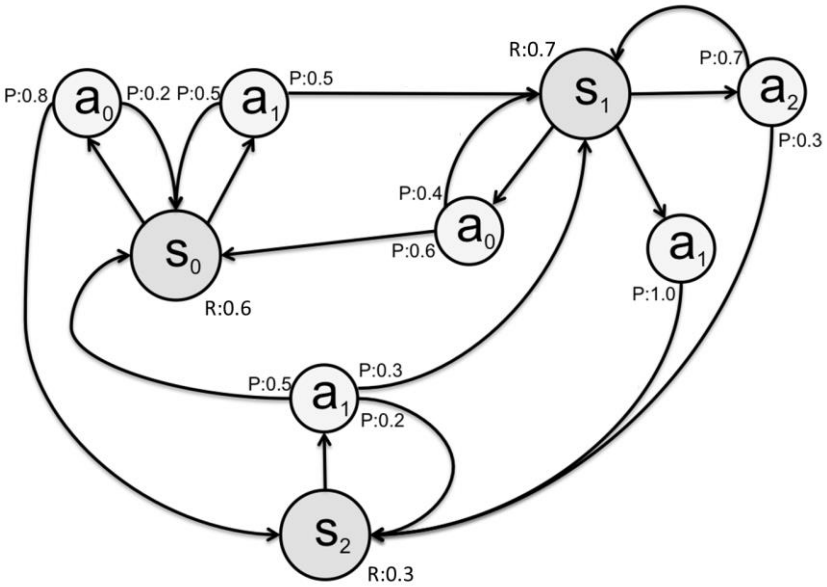

Figure 4 Example of a simple MDP

As an MDP is a stochastic process the outcomes of the actions taken are partly random, and thus not fully under the control of the decision maker. The probability of transitioning to state *s'* is dependent on the current state *s* and the action *a* being performed and is conditionally independent of all previous states and actions taken. This attribute of an MDP transition function is known as the Markov property.

An essential concept in understanding MDPs is that of the policy π. A policy specifies the action that will be chosen to be performed from any given state e.g. $\pi(s)$ represents the action to be taken when in state *s*. A policy can thus be looked upon as one possible solution to the MDP. To solve a problem modeled as an MDP one must find its optimal policy π^*. This optimal policy essentially maps process states to the best possible actions that can be taken from these states i.e. the action with the highest utility from that state.

The utility function $U_\pi(s)$ returns a value that numerically represents the long term reward of being in state *s* and following policy π thereafter. The utility of a sequence of states can be calculated in a number of ways. The first approach is to simply sum the short term rewards of each state within the sequence. Agents that use this approach are said to use additive rewards. The second approach is to use discounted rewards. Discounting rewards is a method of mathematically representing the increasing uncertainty in increasingly distant future rewards. A discount factor γ (between 0 and 1) sets the agent's confidence in current rewards over future rewards. A discount factor of 1 would give full confidence in future rewards (which is essentially equivalent to additive rewards) while a discount factor of 0 gives confidence only to

the most recently received rewards. Equation (1) illustrates discounted rewarding over a sequence of states.

$$U([s_0, s_1, s_2]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_0) + \cdots \quad (1)$$

This leads to the following equation:

$$U([s_0, s_1, s_2]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) = \gamma \sum_{t=0}^{\infty} R(s_t) \quad (2)$$

Taking into account the probabilities associated with the different states s' that can be transferred into, which is essential in stochastic environments, the utility of being in state *s* can now be calculated using equation (3).

$$U_\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U_\pi(s') \quad (3)$$

The optimal policy π^* can be calculated by selecting the action for each state that maximizes the expected utility. This principle is illustrated below:

$$\pi^*(s) = \underset{a \in A(s)}{argmax} \sum_{s'} P(s'|s, a) U(s') \quad (4)$$

This equation combines with equation (3) to lead us to the following method of calculating the optimal action to be taken from state *s*:

$$\pi^*(s) = \underset{a \in A(s)}{argmax} R(s) + \gamma \sum_{s'} P(s'|s, a) U(s') \quad (5)$$

When brought back into equation (3) we get the following optimal policy based utility function:

$$U_{\pi^*}(s) = R(s) + \gamma \underset{a \in A(s)}{max} \sum_{s'} P(s'|s, a) U(s') \quad (6)$$

or

$$U_{\pi^*}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi^*(s)) U(s') \quad (7)$$

This is known as a Bellman equation for utility. For MDPs a Bellman equation (named after the applied mathematician Richard E. Bellman) refers to a recursion of expected rewards.

2.2.2 Learning Algorithms

2.2.2.1 Dynamic Programming

Dynamic Programming (DP) (Sutton & Barto, 1998) refers to a set of algorithms that solve complex problems by breaking them down into simpler sub-problems. Each of these sub-problems is solved and their results are combined to give an overall solution. Each sub-problem can often be repeated multiple times in calculating the final solution. Benefits of this approach over other possible approaches are speed and simplicity. DP is a common approach to calculating the optimal policy of an MDP modeled control problem. In this subsection we will introduce the two main DP approaches used in MDP resolution, namely value iteration and policy iteration. These approaches are based upon DP Bellman equations such as those introduced in the previous section.

2.2.2.1.1 Value Iteration

Value iteration (Russell & Norvig, 2010) algorithms begin by finding the utility of each state within the state space. The simultaneous equations necessary to achieve this goal are however nonlinear and cannot be calculated using linear algebra. The most common alternative is to iteratively apply a Bellman equation to each state until convergence is achieved. Each iterative step is called a Bellman update. This update is performed on all states within the state space simultaneously. The Bellman update based upon equation (7) is given below:

$$U_{t+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_t(s') \tag{8}$$

Equilibrium is guaranteed if the update is applied to the state space an infinite number of times. Once this iterative process has been applied the optimal policy $\pi^*(s)$ can be calculated using equation (5). Assuming a perfect environment model (given by *P(s'|s,a)* and *R(s)* or *R(s'|s)*) this approach guarantees finding the optimal policy. The value iteration algorithm is given below.

Initialize U arbitrarily, e.g. U(s) = 0 for all s ∈ S

repeat

U ← U'; δ ← 0

For each state s ∈ S

$U_{t+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_t(s')$

If$((U_{t+1}(s) - U_{t+1}(s)) > \delta)$

$\delta \leftarrow U_{t+1}(s) - U_t(s)$

until δ *< some small threshold value*

return U

Figure 5 Value iteration algorithm

2.2.2.1.2 Policy Iteration

Although the value iteration algorithm is guaranteed to find the optimal policy π^* it can be quite inefficient because of the fact that not every state needs to have a highly accurate utility value. For example if in one state a specific action is clearly better than the others then the accuracy of the utility value need not be exact. The Policy iteration algorithm (Russell & Norvig, 2010) maximizes on this principal. The Policy iteration algorithm iteratively alternates between two steps to improve a policy to the point that it is optimal. The first step is policy evaluation and the second is policy improvement. The algorithm begins with a random policy. The policy evaluation process evaluates this policy by calculating the utility of each state in the state space using equation (3). Because the policy fixes the action to be selected it turns out that this step is much simpler than solving the regular Bellman equations that need to be solved by the value iteration algorithm. This is because unlike the Bellman equations these are linear and can be solved using standard linear algebra. The policy improvement process then adjusts and improves the policy using hill climbing or one-step look-ahead based on equation (4). The algorithm terminates when the policy improvement process stops making changes to the policy. As shown in equation (9), this process continues iterating until an optimal policy is converged upon.

$$\pi^0 \xrightarrow{E} U_{\pi^0} \xrightarrow{I} \pi^1 \xrightarrow{E} U_{\pi^1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \xrightarrow{E} U_{\pi^*} \tag{9}$$

where:

- $\xrightarrow{E}$ represents a policy evaluation process
- $\xrightarrow{I}$ represents a policy improvement process

The policy iteration algorithm is given below.

Initialize π randomly and U arbitrarily, e.g. U(s) = 0 for all s ∈ S

repeat

// Policy Evaluation

repeat

$\delta \leftarrow 0$

for each state s ∈ S

$u \leftarrow U_\pi(s)$

$U_\pi(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U_\pi(s')$

$\delta \leftarrow \max(\delta, u - U_\pi(s))$

until δ > *some small threshold value*

// Policy Improvement

unchanged ←*true*

for each state s ∈ *S*

if$(\max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s') > \sum_{s'} P(s'|s,\pi(s))U(s'))$

$\pi(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$

unchanged ←*false*

until unchanged

return π

Figure 6 Policy iteration algorithm

2.2.2.1.3 Challenges

DP algorithms have the important quality of being able to guarantee finding the optimal policy π^* and thus solve a problem modeled as an MDP. There are however two main challenges that limit their use in real world applications. The first of these challenges is the fact that DP algorithms require a perfect model of the environment in order to perform their calculations. This environment model takes the form of the transition function *P(s'|s,a)* and reward function *R(s)* or *R(s'|s)*. In practice it can be extremely difficult and often impossible to obtain such an environment model in advance. This is particularly true of environments that are stochastic, dynamic, multi-agent, partially observable, or unknown. The second challenge stems from the fact that DP algorithms base their calculations on iteratively navigating through an environment model i.e. they are model based. Model based algorithms can be much more computationally expensive than alternative approaches. DPs are however very important theoretically. Other approaches to solving MDPs attempt to approximate DP solutions, while not relying on perfect models of the environment and while using less computationally expensive algorithms.

2.2.2.2 Adaptive Dynamic Programming

Adaptive Dynamic Programming (ADP) (Russell & Norvig, 2010) algorithms overcome one of the major challenges of regular DP algorithms by learning the actual environment model. An ADP agent can thus be introduced to an environment with little or no model of the environment and over time it will build up a transition model and an expected reward model. When the model of the environment has been learned the ADP agent can then solve the corresponding MDP using regular dynamic programming

methods. Because the model is learned through observation of the environment over time it can be seen as a dynamic model. This enables it to adapt to dynamic changes within environments that might be stochastic, dynamic, or inhabited by multiple agents. The following algorithm shows how an ADP agent can learn the environment model over time.

increment N(s')
$\mathrm{R}(s') \leftarrow \mathrm{R}(s') + r'$
$R(s') \leftarrow \mathrm{R}(s')/N(s')$
if (s is not null)
    increment N(s,a) and N(s'|s,a)
    for(each s' where P(s'|s,a) is non zero)
        P(s'|s,a) ← N(s'|s,a)/ N(s,a)

Figure 7 Environment model learning algorithm

where:

- $\mathrm{R}(s)$ is the sum of short term rewards received for being in state *s*
- *r* is the short term reward received for being in state *s*
- *N(s)* is the number of times the agent has been in state *s*
- *N(s, a)* is the number of times action *a* was taken from state *s*
- *N(s'|s,a)* is the number of times action *a* led to a transition from state *s* to state *s'*

When traditional representations are used for the state space, such as a lookup table, ADP becomes inept as the state space size increases. This is because as the number of states in the state space increases it becomes less likely that the agent will visit them enough times to estimate an accurate transition probability or expected short term reward. We will discuss methods of addressing this challenge in section 2.2.3.

2.2.2.3 Temporal Difference Learning

Up to this point we have been looking at solving problems modeled as MDPs by using DP algorithms. In summary, DP algorithms iteratively apply a Bellman equation in calculating the utility values for each state of an MDP's state space. They then use these utility values to determine the MDP's optimal policy π^*, and thus solve the MDP. An alternative approach to solving an MDP is by using Temporal Difference (TD) learning algorithms (Sutton & Barto, 1998) (Russell & Norvig, 2010). Instead of iteratively calculating the utility values of each state at each time step TD algorithms update these values using the difference in utilities between successive states. Thus instead of adjusting a state's utility value to agree with all successive states it is adjusted to agree simply with the single next successive state. Over time, as the number of observed transitions increases, the TD utilities resemble more and more to ADP utilities.

Thus the TD utilities approximate ADP utilities. TD utilities require much less calculations per time step but they are slower to converge to their true values than ADP utilities. TD learning is more formally defined in the algorithm below:

$$U_\pi(s) = U_\pi(s) + \propto \left(R(s) + \gamma U_\pi(s') - U_\pi(s)\right) \tag{10}$$

where:

- $\propto$ is the *learning* rate parameter

Notice that the TD algorithm does not require a transition function $P(s'|s,a)$ to perform its updates. TD algorithms are thus referred to as model free algorithms.

There are a number of distinct TD algorithms. We will now describe three of the major ones, namely Q-Learning, SARSA, and Actor-Critic Reinforcement Learning (ACRL).

2.2.2.3.1 Q-Learning

Instead of learning a utility function *U(s)* Q-Learning (Watkins & Dayan, 1992) learns an action-utility function *Q(s,a)*. The stored values are referred to as Q-Values as opposed to utility values. Traditionally these values would be stored in a lookup table referred to as a Q-Table. The two utility based functions are related as shown below.

$$U(s) = \max_{a \in A(s)} Q(s,a) \tag{11}$$

The Q-Learning update rule is given by equation (12).

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma \max_{a' \in A(s')} Q(s',a') - Q(s,a)) \tag{12}$$

This equation is calculated whenever taking action *a* results in a transition from state *s* to state *s'*. This updates a single Q-Value within the Q-Table. As the number of process iterations nears infinity the Q-function Q(s,a) converges to optimality i.e. $Q^*(s,a)$. The optimal policy $\pi^*(s)$ can then be read directly from the Q-Table. In performing this update the Q-Learning algorithm learns relative to the actions that yield the maximum Q-Values i.e. $\max_{a' \in A(s')} Q(s',a')$. Q-Learning thus learns relative to a greedy policy i.e. one that always chooses the action that yields the highest reward, even if the agent is not following a greedy policy. Q-Learning is thus referred to as being an off-policy algorithm. This approach has the benefits of being flexible enough to be able to learn the best actions to take even when being led by a random policy.

2.2.2.3.2 SARSA

The State-Action-Reward-State-Action (SARSA) algorithm (Sutton & Barto, 1998) has an update equation that is quite similar to that of Q-Learning, with the notable exception of the fact that SARSA learns relative to the agent's current policy. Thus SARSA is an on-policy algorithm. This can be clearly seen in SARSA's update algorithm shown below by the lack of the $\max_{a' \in A(s')}$ statement.

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma Q(s', \pi(s')) - Q(s,a)) \quad (13)$$

This rule is applied at the end of the pattern *s*, *a*, *r*, *s'*, *a'*, which explains the algorithm's name. Whereas Q-Learning excels when the policy includes elements of randomness SARSA can learn quicker and more accurately than Q-Learning when the policy does not.

2.2.2.3.3 Actor-Critic Reinforcement Learning

Actor-Critic Reinforcement Learning (ACRL) (Sutton & Barto, 1998) has a more complicated structure than the TD methods heretofore discussed. Q-Learning and SARSA extract the optimal policy from their action utility functions by selecting the actions that have the highest action utilities. ACRL however stores its policy separately from its utility function. The policy is maintained by a component that is called the *Actor*. It is given this title because its function is to select which actions are to be performed. The utility function is maintained by a component that is called the *Critic*. It is given this title because its function is to criticize the actor's actions. ACRL thus has the flexibility of being able to implement actor and critic using separate technologies, such as fuzzy logic and neural networks. Like SARSA, ACRL is an on-policy algorithm. ACRL's on-policy approach is necessary so that the critic can learn about and critique the policy that the actor is currently using. The critique is represented by a TD error. This TD error is then used to update the utility function, as is done with the other TD algorithms that we have looked at. ACRL also uses the TD error to directly update the policy i.e. the actor, as can be seen in Figure 8.

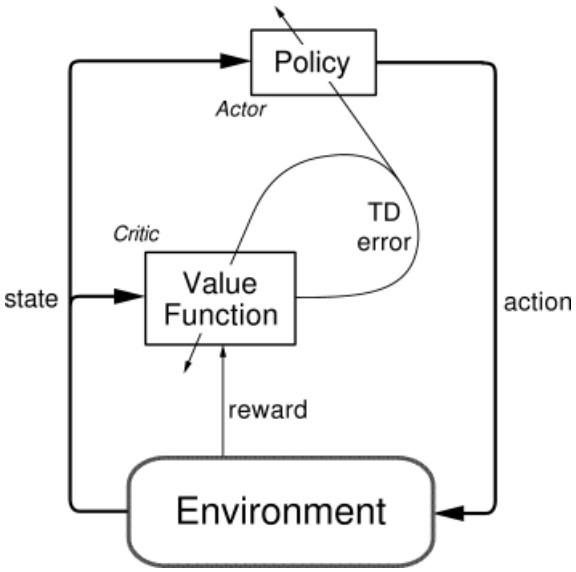

Figure 8 Actor Critic Reinforcement Learning (ACRL) architecture (Sutton & Barto, 1998)

Following the performance of an action the critic uses equation (14) to calculate the TD error, thus determining if the executed action had better or worse consequences than expected.

$$\delta_t = r_{t+1} + \gamma U(s_{t+1}) - U(s_t) \tag{14}$$

where:

- δ_t is the TD error at time step *t*

The actor's preference for selecting that action in the future is then reinforced by that TD error value by using an equation similar to that given by equation (15).

$$P(s_t, a_t) = P(s_t, a_t) + \beta\delta_t \tag{15}$$

where:

- β is a positive parameter similar to learning rate

One major benefit of ACRL is that of its low cost action selection. In systems where action values are continuous, such as robotics for example, the possible set of specific actions to choose from may be infinite. Other TD algorithms would have to calculate an action-utility value for each action within this set before making a selection. These efforts can be avoided if the policy is stored explicitly.

2.2.3 Policy Definition

Solving a problem modeled as an MDP is achieved by discovering which actions should be selected for execution from any state within the state space so as to maximize the agent's utility or long term reward i.e. by discovering the MDP's optimal policy π^*. A policy can be defined in various different ways depending upon characteristics of the actions that can be taken and also upon characteristics of the reward being optimized. In this section we discuss some of these aspects of policy definition that are of relevance to this thesis.

2.2.3.1 Policy Representation

In anticipation to section 2.2.6's discussion on techniques that can be used to represent a policy this section addresses the question of whether or not the policy should be explicitly represented in the first place. The following two subsections discuss implied and explicit policies and factors that would give preference to one being used as opposed to the other.

2.2.3.1.1 Implicit Policies

An implied policy is one that is read directly from the utility or action-utility function (Russell & Norvig, 2010). For example, when a traditional Q-Learner agent needs to know the Q-Value for taking a particular action from a given state it reads the value directly from its Q-Table. It first finds the Q-Table row that corresponds to the given state and then finds the Q-Table column that corresponds to the specific action. The Q-Value is stored in the cell at the intersection of this row and column. Although the policy is a distinct component of this type of agent it is clear to see that it is an implied component, as opposed to being explicitly represented. Agents that have action spaces consisting of very few actions that each have a discreet set of possible action values are often very good candidates for this approach. In the case of traffic control an example of this would be an intersection agent whose action space consisted of increasing or decreasing the phase lengths of any of its phases by a specific number of seconds e.g. by five seconds. If the intersection agent has only four phases then it has a total choice of at most eight possible actions from any state.

A problem can begin to arise however when the number of actions that can be taken grows and when the action values become more granular or even continuous. In our traffic control example this might be the equivalent of a larger intersection agent that has ten phases and that can either increase or decrease the phase length by three, five, seven, ten, thirteen, or fifteen seconds. The agent now has a choice of at most one hundred and twenty possible actions from any one state. The number of possible choices further increases when it is possible for the agent to modify its offset or cycle length or when multiple actions can be taken during a single time step i.e. joint actions. When selecting the optimal action to be performed from a given state the agent must look up the Q-Value for each of the possible actions that can

be taken from that state and select the action associated with the highest value. With utility or action-utility functions being represented using function approximators, which will be discussed in section 2.2.4, more complex calculations than a simple lookup are required for each value to be retrieved. With increasing action space size this approach becomes impractical. One possible solution to this challenge is to cut down on the number of possible actions or action values within the action space, thus reducing its dimensionality and size. When done manually this requires expertise, experience, time, costs, etc. It also has the drawback of providing less flexibility than the system designer may have originally hoped for. Another solution is to use explicit policy representation, which will be discussed in the following subsection. Value iteration, Q-Learning, and SARSA each use implicit policies.

2.2.3.1.2 Explicit Policies

As the number of actions that can be taken grows and as the granularity of their action values approaches continuous levels it can become necessary to explicitly represent the policy. An explicit policy can be represented similarly to the utility or action-utility function, or even the environment model (Russell & Norvig, 2010). The different ways in which these can be represented will be discussed in section 2.2.4. In a simple and traditional approach a policy can be represented using a lookup table that contains a row corresponding to each possible state in the state space and then one single column. The cells of this table contain representations of the actions to be taken from the associated states. Each cell contains a single action representation that has a single action value. During the action selection process only one lookup, or calculation, is required as opposed to the potentially infinite number required using the implicit policy approach. This is regardless of the number of actions that can be taken or of their action value granularity. Agents that implement explicit policies thus have the reflexive agent benefits of rapid action selection. The policy iteration DP algorithm and the ACRL TD algorithm are examples of algorithms that we have looked at so far that explicitly represent their policies.

2.2.3.1.2.1 Local Search Algorithms

Having an explicit policy in place the challenge is then to discover which actions and action values should be contained within the policy. This can be looked upon as a local search problem within the action space (Russell & Norvig, 2010). Local search algorithms that can be used to solve such a problem include methods inspired by statistical physics (simulated annealing) and evolutionary biology (genetic algorithms). Local search algorithms operate using a single current solution, or policy in our case, and generally move only to the neighbors of that solution in their search of the optimal solution. These approaches also have the advantage of very small memory requirements as well as being able to find a reasonable estimate of the optimal policy within a potentially infinite action space.

2.2.3.1.2.1.1 Hill-Climbing Algorithm

One of the more basic local search algorithms is called the hill-climbing algorithm (Russell & Norvig, 2010). To better explain how it works we will look at finding the optimal action for a single state, although in practice the process that we describe will be applied to all states represented within the policy simultaneously. The current policy is randomly initialized so a random action *a* and random action value *v* is assigned to the entry that represents state *s* within the policy. When the agent finds itself in state *s* it evaluates this random assignment a^v by executing the corresponding action *a* with the specified action value *v*. This evaluation of the assigned action a^v may be performed over a number of time steps to take into account the stochastic and dynamic nature of the environment. The policy is then modified by slightly altering the action value *v*. This new assignment a^v is then evaluated and further modified, leading into an iterative evaluation-modification process. As this process progresses the agent begins to get a view of the one dimensional action-value landscape for action *a* taken from state *s*. This landscape is represented below:

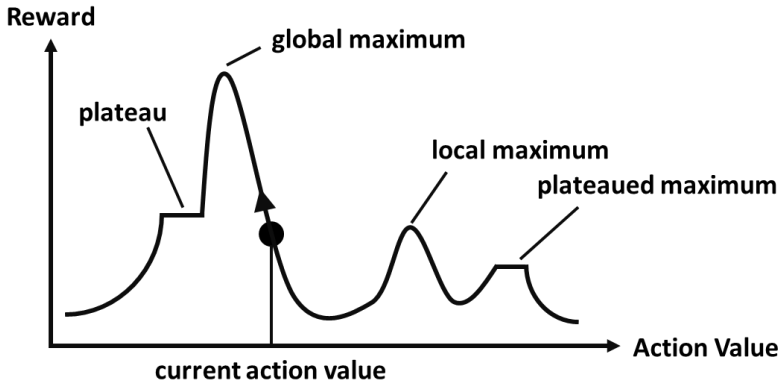

Figure 9 One dimensional action-value landscape for action *a* taken from state *s*

The aim of the algorithm is to find the global maximum within this landscape for action *a* in state *s*. Thus the modifications made to action value *v* should always be in the direction of the global maximum. The algorithm however does not have a complete view of the landscape but only has a local view of the current action value and values immediately adjacent to it. Thus the algorithm can observe the current slope but not whether or not it is at the highest point within the landscape. Constantly adjusting the action value in the direction of increasing reward i.e. uphill, is what gives the hill-climbing search algorithm its name. This algorithm stops when a peak is reached. The hill-climbing algorithm is the inverse of the gradient descent search algorithm which aims at finding the local minimum in a value landscape.

2.2.3.1.2.1.2 Simulated Annealing

The hill-climbing algorithm is however unfortunately susceptible to getting stuck on local maxima or on plateaus. In order to overcome this drawback a technique called simulated annealing may be used

(Russell & Norvig, 2010). In materials science annealing refers to a process of heating a material to a very high temperature and then gradually cooling it so as to alter its physical or chemical properties. Whereas hill-climbing always chooses to adjust the action value in the direction of increasing reward the simulated annealing algorithm always chooses to adjust the action value in a random direction. If the chosen direction is uphill then the adjustment will be made with a probability of 1 i.e. it will be made. If however the chosen direction is downhill and will decrease the reward then the adjustment will be made with some probability less than 1. This probability decreases exponentially with the expected decrease in expected reward ΔE i.e. it is less likely to descend steep slopes. The probability also decreases as the temperature variable T decreases. T starts with a high value close to 1 and decreases over time. If T decreases slowly enough then the algorithm becomes much more likely to find the global maximum. The simulated annealing algorithm is given below:

```
Current ← random value
for( t ← 1; ∞; t++)
    T ← timeReduction(t)
    if(T == 0)
        return Current
    Next ← Current randomly + or - Step
    ΔE ← reward(Next) - reward(Current)
    if(ΔE > 0)
        Current ← Next
    else
        Current ← Next with probability e^(ΔE/T)
```

Figure 10 Simulated annealing algorithm

where:

- $Current$ and $Next$ are the current and next action values respectively
- Step is a random step size by which the action value is incremented or decremented

Up to this point we have assumed that although the action value *v* is modified over time the actual random action *a* that was chosen during initialization remains the same. During each time step however a new random action should be selected with a probability less than *T*.

An approach named local beam search is similar to simulated annealing with the exception that it keeps track of a number (*k*) of policies instead of just one. These policies communicate their results with each other so that they all begin to converge to the most fruitful area of the action space.

2.2.3.1.2.1.3 Genetic Algorithms

Genetic Algorithms (GA) (Russell & Norvig, 2010) maintain a population of *k* policies similar to the local beam search algorithm. With the algorithms described so far we have said that a policy is modified by changing its actions and action values. This can also be looked upon as a new policy being generated from an evaluated one and then the evaluated one being discarded. GAs generate new policies based upon parent pairs of evaluated policies. Policies that perform better have a higher chance of generating child policies i.e. a better chance of procreation. Thus GAs allow for the survival of the fittest policy in the search for the most fit solution i.e. the optimal policy. For two policies to combine a random percentage *P* of the action values of one (which are looked on as being analogous to chromosomes within a parent's genes) and 1-*P* percent of the action values of the other parent policy are used in the initialization of the offspring policy. This random percentage is referred to as the crossover point. Some random action values are also inserted into the child policy to allow for mutation and variance within the population.

2.2.3.2 Multiple Policies

In defining a policy one must also take into consideration characteristics of the reward being optimized. Of particular interest to the discussion within this section is the number of rewards being optimized. Whereas RL is traditionally a single-policy algorithm i.e. one that optimizes on a single reward, there have been numerous multi-policy extensions to allow for optimization across a number of rewards. To solidify this concept we will apply it to a traffic control scenario. A common approach to reward calculation for an intersection agent is to use either vehicle throughput, waiting time, or queue length, or some combination of two or more of these. This works fine when all vehicles are equal, but what is to be done when public transport vehicles like busses are to be given a higher priority? In this situation two rewards can be potentially calculated at each time step, one based on vehicles in general and the other based on public transport vehicles. This could lead to either a single combined policy or two separate policies (multi-policy). The question to be addressed in this section is how multiple policies can be handled by learning agents.

2.2.3.2.1 Combined Policy

One approach to handling multi-policy optimization is to combine all policies into one (Barrett & Narayanan, 2008). This single policy is then a product of the others. Within this product policy each of the component policies can be assigned a weight in accordance with their importance. In our traffic control example a single public transport vehicle could be given a weight that would make it the equivalent of ten regular vehicles, thus giving a much higher priority to approaches that have public transport vehicles on them as they return much higher rewards. The weights assigned to the different rewards can remain constant (Gábor, Kalmár, & Szepesvári, 1998) or they can change over time (Natarajan & Tadepalli, 2005). It is also possible to navigate a policy subspace through consideration of

all combinations of reward weights (Hiraoka, Yoshida, & Mishima, 2008), however this approach must be able to handle state space explosion appropriately. Combining multiple policies into one is an approach that can be quite effective when the number of policies being combined is relatively small. This approach however does not scale well as adding policies to the combination increases the learning time exponentially (Cuayáhuitl, Renals, Lemon, & Shimodaira, 2006).

2.2.3.2.2 Arbitration Based Approaches

A second approach to handling multi-policy optimization is to have each policy represented separately and then have some form of arbitrator decide which action to choose based upon the suggestions of these policies and weights assigned to them. The arbitrator could either choose one of the suggested actions (Brooks, 1991) (Gadanho & Hallam, 2001) or perhaps even a compromise action (Rosenblatt, 2000) (Russell & Zimdars, 2003).

2.2.3.2.2.1 W-Learning

W-learning (Humphrys, 1998) is an arbitration based approach which learns the weights to be assigned to each policy in each state as opposed to relying on manual predefinition of the weights. W-learning maintains a separate Q-Learning process for each implicit policy. At each time step each of these policies suggests which action should be taken. Each Q-Learning process then observes the resulting reward, regardless of whether or not its suggested action was executed, thus learning what happens when its suggestions are either executed or ignored. Based upon these observations the agent can learn a weight, or W-value, that reflects the importance of the suggested actions for each policy from each state. These W-values are learned relative to the other policies simultaneously deployed and are thus not absolute values. For example a policy's W-value for a particular state could be relatively high if its suggested action is in conflict with the actions suggested by the other policies, or it could be relatively low if the actions suggested by some of the other policies are as suitable as its own suggested action. The W-value update equation is given below:

$$W_i(s) = (1-\alpha)W_i(s) + \alpha(Q_i(s,a_i) - (R(s) + \gamma \max_{a_i' \in A_i(s')} Q_i(s',a_i'))) \quad (16)$$

where:

- a_i is the action suggested by policy i

At each time step, once each of the policies has suggested which action should be performed the action to be actually executed can be selected using one of a number of different individual or group methods. Individual methods select which action to execute based on some criteria of a single policy, for example the policy could be selected that has the highest W-value for that state and thus the one that is going to suffer the most if its suggested action is not performed (Humphrys, 1998). Group methods take

into account all policies, for example the action that satisfies the most amount of policies will be selected for execution. The rewards returned to each policy should be designed in such a way so as to reflect the policy's priority i.e. a policy of higher priority should receive higher rewards than one of lower priority. When this is observed policies with higher priorities will be given a higher preference during action selection time due to their magnified W-values. Policies of lower priority will however still have a chance of having their actions selected during action selection time if their actions are of high importance to them while policies of higher priority are in a state for which they have assigned a low importance. Only the policies whose action suggestions were not performed will have their W-values updated so as to ensure that no policy gets neglected for an extended period of time. Dusparic et al. (Dusparic & Cahill, 2010) have performed a significant volume of research with a focus on applying W-learning to traffic control.

2.2.4 Representation

In the previous subsections we have looked at two main approaches to learning, i.e. DP and TD learning (Russell & Norvig, 2010). Where the former learns an environment model and a utility function the latter learns an action-utility function with no environment model. The question now arises as to how these functions, as well as an explicit policy, can be represented. In this section we will look at a range of function representations ranging from the traditional lookup table to other parametric and non-parametric approaches. Every function representation in RL that is not a lookup table is referred to as a function approximator.

2.2.4.1 Lookup Tables

Traditionally utility functions and action-utility functions were assumed to be represented in tabular form referred to as a lookup table (Russell & Norvig, 2010), or specifically as a Q-Table in the case of Q-Learning (Watkins & Dayan, 1992). Using a Q-Table for example, the Q-Value of performing a specific action from a given state is determined by finding the Q-Table row that corresponds to the given state, finding the Q-Table column that corresponds to the specific action, and then finding the Q-Value that is stored in the cell at the intersection of the two. This approach thus has the benefit of simplicity. With only one state variable a lookup table can be represented as a simple two dimensional table as illustrated in the example below.

States \ Actions	Extend Phase Length	Maintain Phase Length
Vehicles Present	0.9	0.2
Vehicles Absent	0.2	0.8

Table 1. Lookup table example

In this example of basic actuated traffic control (see section 2.3.2.2) we can see that when vehicles are present the agent can achieve a utility of 0.9 if it extends the phase length but only a utility of 0.2 if it maintains the phase length. In this example there is only one state variable i.e. whether vehicles are present or absent. The state however may not be that simple in practice e.g. it might be important to know which approach of the intersection the vehicles are present on and how many vehicles are present. In this case more state variables must be added to the table, and thus more dimensions. Our sample state variable (vehicle presence) represents a boolean variable, yet some state variables may have multiple values (discrete state variables) or even an infinite number of values (continuous state variables). As the number of dimensions as well as their sizes increase the size of the Q-Table increases exponentially. As the size of the Q-Table increases so does the length of time taken to populate it i.e. to learn, and in many instances it could even take an infinite amount of time. As each state needs to be visited an infinite number of times in order to converge to the correct utility value or Q-Value we can see that lookup tables soon become unable to achieve their purpose. The traditional approach to addressing this issue is to be wise in the definition of state variables with regards to their number and size. Thus keeping the lookup table size as small as possible. In situations where this is possible the agent may continue to benefit from the simplicity of lookup tables. Ponsen et al. (Ponsen et al., 2010) propose an abstraction technique that reduces the size of a state space by filtering out irrelevant state variables. Ponsen et al. also describe a generalization technique in that same study, which allows an agent to apply knowledge from one part of the state space to other, similar yet separate parts of the state space. These concepts have recently received significant attention in the machine learning research community. Although abstraction increases speed and efficiency in rooting out the least significant state variables it does not take into account the fact that some state variables are only significant under certain conditions e.g. an intersection might be heavily dependent on a neighboring downstream intersection during periods of extremely high congestion and not at all dependent on it during periods of low congestion.

2.2.4.2 Function Approximation

In many instances it is simply not possible to keep a lookup table small enough to allow it to perform efficiently. In these instances other function representations need be investigated. In the context of agent learning any function representation besides of a lookup table is referred to as a function approximator (Sutton & Barto, 1998). A key benefit of function approximation is that it facilitates the concise representation of utility functions that have very large state spaces. Another extremely important benefit of function approximation is that it allows a learning agent to generalize from states it has visited to states that it has not visited. The values returned by a function approximator are however only an approximation to the true value that might be obtained using a lookup table over an infinite amount of time. Also, as lookup tables were originally used in the development of learning algorithms, such as the DP and TD

algorithms that we have looked at so far, these algorithms tend to require modification in order to cope with the integration of function approximators. Papavassiliou and Russell (Papavassiliou and Russell, 1999) alternatively present a different type of reinforcement learning that converges using any form of function approximation, so long as a best-fit approximation can be found for the underlying data. Function approximation is also very useful for learning and representing environment models for algorithms such as ADP. In this section we will look at a number of different parametric and non-parametric approaches to function approximation.

2.2.4.2.1 Parametric Models

A parametric function approximator is a learning model that summarizes data with a fixed set of parameters. The number of parameters required to represent the underlying data does not change regardless of the amount of data points that have been used to train it. It will thus maintain its original size regardless of how much data it represents.

2.2.4.2.1.1 Linear Models

Many state spaces can be represented using simple linear functions (Russell & Norvig, 2010), as in the illustration below:

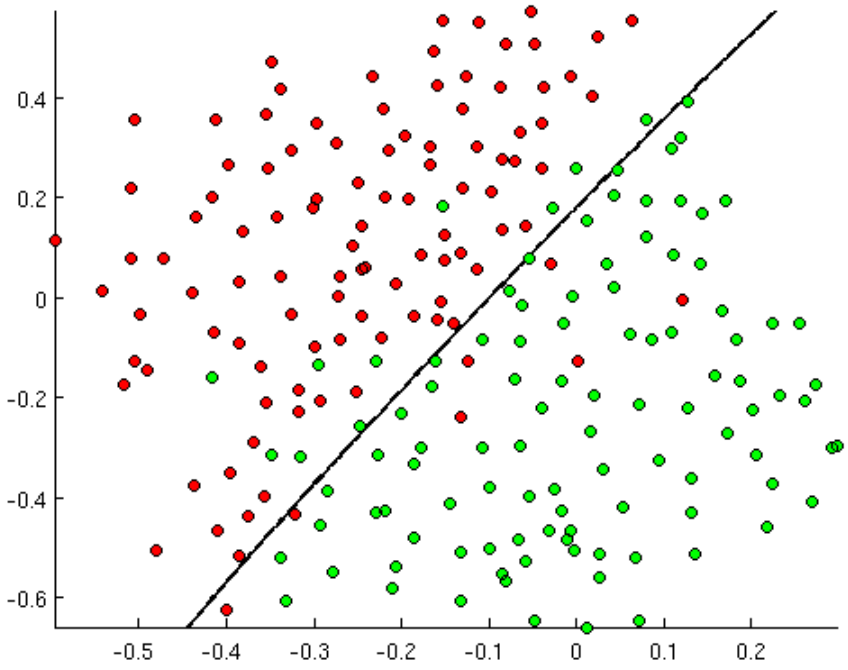

Figure 11 Linearly separable state space (Stanford OpenClassroom accessed 06/04/2015)

When this is the case the approximate utility of being in state s $\bar{U}(s)$ can be calculated with the state variable values $s_1 - s_N$ (***N*** representing the total number of state variables) and the parameters $\theta_0 - \theta_N$ using the following equation:

$$\bar{U}_\theta(s) = \theta_0 + \theta_1 s_1 + \theta_n s_n + \cdots \theta_N s_N \tag{17}$$

At each time step in the learning process we update the parameters by reducing the temporal difference between successive states. This can be done by drawing upon the TD and Q-Learning equations given as (10) and (12) to define the following parameter update equations.

$$\theta_n = \theta_n + \propto \left(R(s) + \gamma \bar{U}_\theta(s') - \bar{U}_\theta(s)\right) \frac{\vartheta \bar{U}_\theta(s)}{\vartheta \theta_n} \tag{18}$$

$$\theta_n = \theta_n + \propto \left(R(s) + \gamma \max_{a' \in A(s')} \bar{Q}_\theta(s', a') - \bar{Q}_\theta(s, a)\right) \frac{\vartheta \bar{Q}_\theta(s, a)}{\vartheta \theta_n} \tag{19}$$

where:

- $\frac{\vartheta \bar{U}_\theta(s,a)}{\vartheta \theta_n}$ is the rate of change of the error with respect to parameter θ_n

A number of studies have demonstrated the use of linear function approximators for DP and TD learning (Sutton, 1988) (Dayan, 1992) (Tsitsiklis and Van Roy, 1997).

2.2.4.2.1.2 Artificial Neural Networks

Many state spaces cannot be linearly divided, as illustrated below, and can thus be modeled using non-linear techniques (Russell & Norvig, 2010).

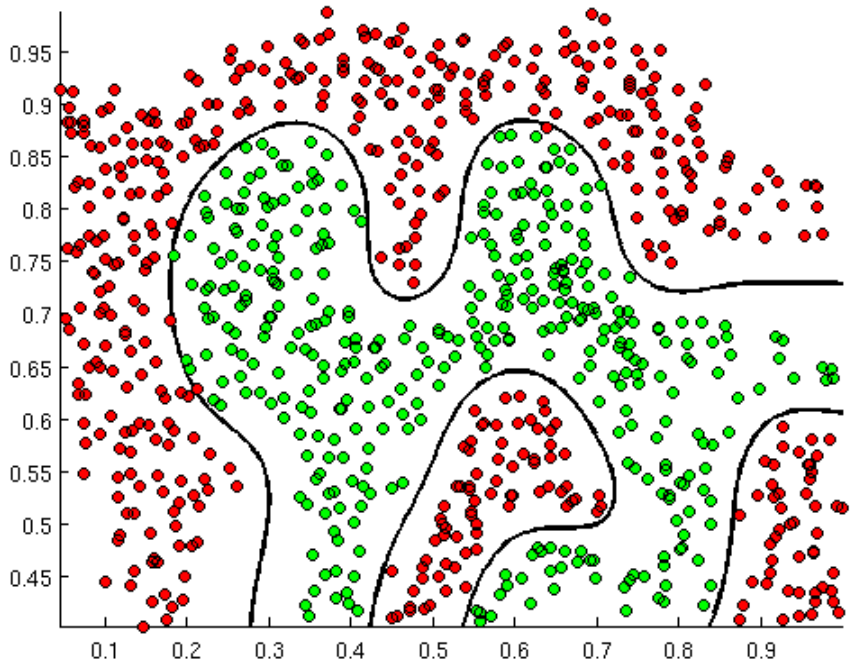

Figure 12 Non-linearly separable state space (Stanford OpenClassroom accessed on 06/04/2015)

A classic example of this is the XOR operator, whose state space, unlike other operators like AND and OR, cannot be linearly divided, as can be seen in the following illustration:

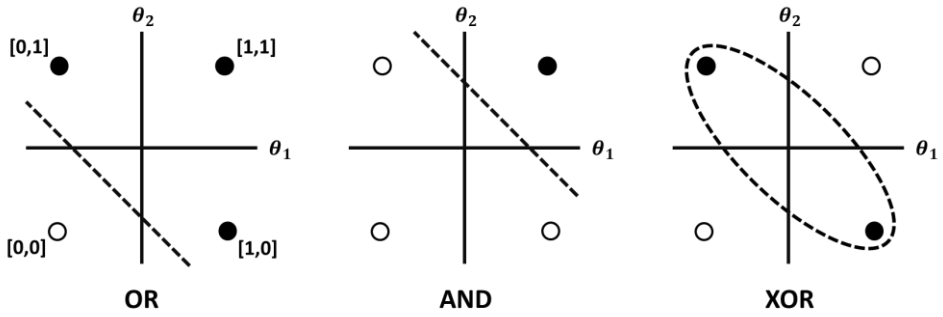

Figure 13 The (non)linear separability of the OR, AND, and XOR operators

Artificial Neural Networks (ANNs) (Russell & Norvig, 2010) are an excellent example of non-linear parametric algorithms that are commonly used for function approximation. ANNs are composed of groups of highly interconnected neurons, which are programmed software components that simulate the

properties of biological neurons. These artificial neurons learn over time by adjusting the weights associated with the connections between them. In a multi-layer ANN the first layer, or input layer, contains an input node for each state variable. The network also contains an inner hidden layer, or possibly multiple hidden layers, that consist of multiple hidden artificial neurons. The final layer, or output layer, contains one output node for each output value. Each node in a layer *i* within the ANN is directly connected to each node in the next layer *j*, thus each input node I_n is directly connected to each node H_m within the hidden layer, and each node H_m within the hidden layer is directly connected to each output node O_k. Each connection has a numeric weight w_j^i associated with it that determines the strength of the connection. Each node within the hidden and output layers also have a bias input (with a constant value of 1) with an associated bias weight w_j^0. This ANN architecture is illustrated below:

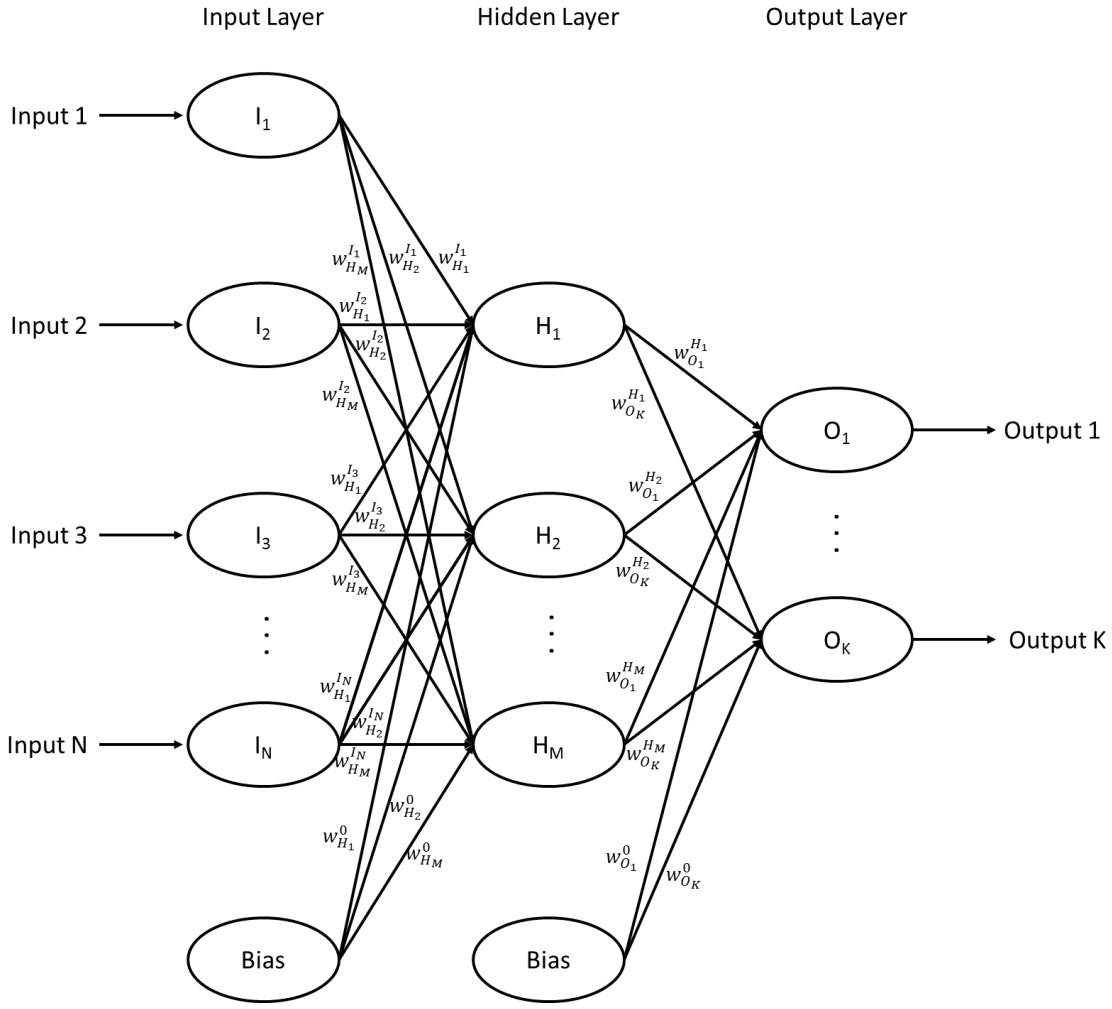

Figure 14 Artificial Neural Network (ANN) architecture

Each node within the ANN receives as input *a* the output values of the nodes within the preceding layer. On receiving these values each neuron calculates the weighted sum *in* of values using the following equation:

$$in_j = \sum_{i=0}^{M\ or\ K} w_j^i a_i \tag{20}$$

Each node then applies an activation function g to this sum *in* to derive the node output i.e. $a_j = g(in_j)$. This activation function is usually a sigmoid function, or in the case of the network being a simple perceptron a basic threshold approach is used. This process of calculating the node output is illustrated below:

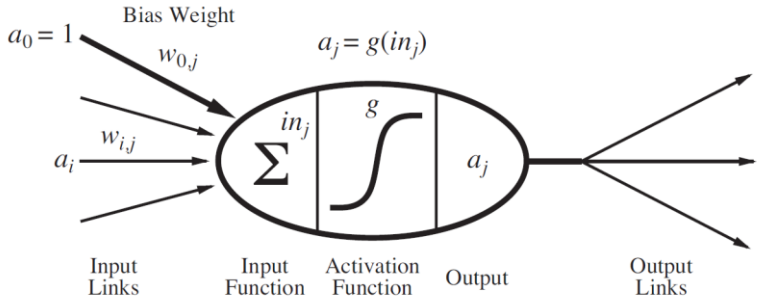

Figure 15 Basic mathematical model of a neuron (Russell & Norvig, 2010, p728)

In a feed-forward ANN training is achieved by generating an error value each time the network is used and back propagating it through the network to adjust the weights, thus either strengthening or weakening the neural connections. The first step in back propagating this error value is to calculate the individual error a.k.a. the delta Δ, of each output node using the following equation:

$$\Delta_j = g(in_j)(y_j - a_j) \tag{21}$$

where:

- y_j is the value that a_j should have been

The delta Δ for each node in the hidden layer can now be calculated using the following equation:

$$\Delta_i = g(in_i) \sum_j w_j^i \Delta_j \tag{22}$$

Once these delta values have been calculated the ANN weights can be updated using the following update rule:

$$w_j^i = w_j^i + \alpha a_i \Delta_j \tag{23}$$

where:

- α is the learning rate [0…1]

The back propagation ANN algorithm is given below.

repeat
    for(each weight w_j^i *in network)*
        w_j^i *= a small random number*
    for(each datapoint in training set)
        // Forward propagation of inputs to compute outputs
        for(each node i in input layer)
            $a_i = input_i$
        for(each layer l other than the input layer)
            for(each node j in layer l)
                $in_j = \sum_i w_j^i a_i$
                $a_j = g(in_j)$
        // Back propagate error to update weights
        for(each node j in output layer)
            $\Delta_j = g(in_j)(y_j - a_j)$
        for(each hidden layer l)
            for(each node i in layer l)
                $\Delta_i = g(in_i) \sum_j w_j^i \Delta_j$
        for(each weight w_j^i *in network)*
            $w_j^i = w_j^i + \alpha a_i \Delta_j$
until termination criteria is met

Figure 16 Back propagation ANN learning algorithm

One very important aspect of ANNs is having the correct network structure i.e. the number of hidden layers and the number of hidden nodes within each layer. If a network is too large for its purposes then it will become over-fitted to the training data and will not be able to generalize well to inputs that it has not

seen before. A common approach in choosing the best network structure is to create a number of various network structures and then keeping the best one.

A very well-known example of an ANN being used as a function approximator is TD-Gammon (Tesauro, 1992, 1995). The use of ANNs as function approximators is however regarded as somewhat of a delicate art. One of the major deterrents in using an ANN function approximator is the complexity of integrating it into traditional lookup table based Reinforcement Learning algorithms. Another significant issue is that its network structure is quite static after its initial training phase, whereas as an agent learns over time using DP or TD algorithms the state space may change significantly, requiring a more dynamic network structure. ANN-based TD learners also tend to forget earlier experiences, particularly those that occur in areas of the state space that are not often visited again once the ANN has been trained. This can result in recurring failures when the agent finds itself in these uncommon states. Instance based function approximators can help avoid this issue (Ormoneit and Sen, 2002) (Forbes, 2002).

2.2.4.2.2 Non-Parametric Models

Unlike the parametric approach, non-parametric function approximation (Russell & Norvig, 2010) does not use a fixed set of parameters to represent the utility or action-utility function. This leads to a correlation between the state space size and the size of the model. For example, data points representing each state ever visited could be stored within the model and then utilized in making utility estimations on subsequent states. Thus the size of the model grows with the number of states visited. This approach to learning is similar to the idea of a lookup table, with the exception that it contains additional functionality that enables the generalization required to deal with large state spaces.

2.2.4.2.2.1 Clustering

Many clustering algorithms exist that can improve upon the lookup table approach by giving it generalization functionality. Some such models include: centroid models e.g. k-means algorithm (Lloyd, 1982), connectivity models (Karypis et al., 1999), and density models e.g. DBSCAN (Ester, Kriegel, Sander, & Xu, 1996). We will look at the k-nearest neighbor clustering algorithm as an example. When an agent is in state s_0 this clustering algorithm firstly retrieves the stored utility values that were received when the agent was in the states $s_{1\text{-}K}$ most similar to s_0 in the past. Similarity between states is usually measured with a Minkowski distance L_p defined below:

$$L_p(s_0, s_k) = \left(\sum_i \left(s_0^i - s_k^i\right)^p\right)^{1/p} \tag{24}$$

where:

- *i* is a state variable

- p is a specified variable

With p equal to 2 this equation gives the Euclidean distance (straight line between the two points) while with p equal to 1 it gives the Manhattan distance (sum of straight lines along the axes between two points e.g. Travelling between two points in Manhattan can typically not be done by following a single straight line but must take into account street corners within the transport grid). Having retrieved the utility values that were received in the K states i.e. $s_{1\text{-}K}$, that are most similar to state s_0 i.e. its K nearest neighbors, the algorithm can now calculate their mean, median, or some other calculable value as an estimate of the utility that will be received when in state s_0.

As the number of state variables taken into consideration increases however a number of issues become apparent (Kriegel, Kröger, & Zimek, 2009), namely:

- The state space size increases exponentially i.e. the curse of dimensionality
- The concept of distance i.e. similarity, between two data points becomes meaningless
- The relevance of certain state variables may differ in different clusters a.k.a. the local feature relevance problem
- Clusters may exist in arbitrarily oriented affine subspaces

Multiple algorithms exist that attempt to perform clustering with high dimensional data. Among the more popular methods are subspace clustering and projected clustering. Subspace clustering algorithms e.g. SUBCLU (Kailing, Kriegel, & Kröger, 2004) detects clusters that exist within all possible subspaces within the state space, each subspace being made up of a unique combination of state variables. Projected clustering algorithms e.g. PreDeCon (Bohm, Railing, Kriegel, & Kroger, 2004) use a distance function to amplify the distance along dimensions of less relevant variables. Once the distance function has been applied then other clustering algorithms e.g. DBSCAN can be used for cluster detection.

2.2.4.2.2.2 Locality-Sensitive Hashing

A hash table is a data structure that implements an associative array, which is a structure that can map keys to values. It uses a hash function to compute an index into an array of buckets from which a desired value can be found. Hash tables have the potential to provide much faster lookup than regular lookup tables or even binary trees. Hash tables rely on exact matching as the allocation of bin space is done randomly e.g. a bin containing a utility value for state s_0 will not necessarily be located anywhere close to the bin containing a utility value for state s_1, which is the nearest neighbor of state s_0. To allow for generalization however a function approximator needs to be able to find states that are similar to the current state, requiring some form of organized storage. Thus in order to use hash mapping for function approximation states that are similar to each other have to group their utility values into the same bin, making the hash function locality-sensitive. Locality-Sensitive Hashing (LSH) (Gionis et al., 1999) (Russell & Norvig, 2010) rapidly retrieves the approximate set of nearest neighbors s'_{1-K} to state s_0. In order to accomplish this LSH requires a hash function $g(s)$ that for any two states s and s' the probability

is high that they will be assigned the same hash key if they are within distance *cr* of each other, where *c* is a specified parameter for the algorithm and *r* is a specified threshold radius value. Similarly, the probability is low that they will be assigned the same hash key if they are not within distance *cr* of each other. This hash function *g(s)* relies on the concept that if two states within an n-dimensional state space are similar then they will also be similar when projected down onto a one-dimensional state space i.e. a line. Thus all state variables in the state space are collapsed except a small subset, which subset is used to judge the distance or similarity between the two states. Although this groups states together that are similar to each other there will also be multiple states included into bins that are quite dissimilar with regards to some of the collapsed states. Thus in one dimension the states will be close together yet in another they may be quite far apart. To increase the probability of finding the actual nearest neighbor states s_{1-K} multiple such hash tables are maintained, each one being projected onto a different random subset of state variables. When the agent is in state s_0 LSH uses s_0's hash key in looking up the associated bucket from each of the hash tables being maintained. The nearest neighbor states $s_{1\text{-}K}$ have a high probability of being contained within all of the referenced bins. The actual distances are calculated between state s_0 and each of the states within the union subset of states contained within these buckets, thus leading LSH to the approximate set of nearest neighbors s'_{1-K}. One excellent example of LSH being used in a real world problem is Torralba et al.'s use of LSH in finding nearest neighbor images within a data set of 13 million web images (Torralba et al., 2008). LSH only had to examine a few thousand images before finding the nearest neighbor images, thus speeding up the search a thousand fold when compared to other algorithms.

2.2.4.3 Conclusion

In this section we have described a number of methods that can be used in representing utility functions, action-utility functions, environment models, and even explicit policies. These methods define the learning capabilities of such components. The second of the two research questions that we address in this thesis (RQ2) focuses on the detection of a method of representing an RL agent's utility function that is well suited to the non-static nature of a dynamic transport network. This utility function representation must be able to perform generalization and abstraction so as to increase efficiency and reduce learning times. This enables the learning agent to learn effectively in such an environment. The utility function representation must also be intuitive with regards to RL equations such that they can be implemented using this representation without requiring major modification themselves. In this section we have looked at a range of function representations ranging from the traditional lookup table to other parametric and non-parametric approaches. Every function representation in RL that is not a lookup table is referred to as a function approximator. The question remains as to the suitability of each of these methods as adequate

solutions to RQ2. In order to assess the suitability of each approach we have composed the following table, which we will now discuss in more detail.

Method	Generalization	Abstraction	Memory Requirements	Processing Requirements	Integration Complexity	Has Been Applied to RL
Lookup Table	No	No	Low	Low	-	Yes
Linear Model	Yes	No	Low	Low	Complex	Yes
Artificial Neural Network	Yes	Yes	Low	Low	Complex	Yes
Cluster Space	Yes	Yes	High	High	Medium	Yes
Locality Sensitive Hashing	Yes	Yes	Low/Medium	Low	-	No

Table 2. Representation method comparison

In Table 2 we can observe the following information for each method of representation.

- Their ability to perform generalization and abstraction as described in this section
- Their memory and processing requirements relative to each other
- The relative complexity of the necessary RL equation modifications
- Whether the methods have been applied to RL within current literature

Traditionally utility functions were designed so as to be represented by lookup tables. These tables do not provide generalization or abstraction functionality. This means that they would not be a suitable method in environments that are represented by high dimensional state spaces. In these situations learning would simply take too long. Lookup tables do however have a number of benefits. They do not take up much space in memory and they have low processing requirements. RL equations have been initially designed to work with lookup tables and thus the two are integrated by design.

Using linear models allows for generalization at the cost of complex changes being required in the RL equations. Linear models do not however facilitate abstraction in learning. Linear models are also quite limited in the data that they represent due to the fact that they represent regions within the state space using straight lines or planes, and can thus not identify non-convex regions. This drawback could be overcome by using Support Vector Machines as a linear model, at the cost of even further complex RL equation modifications. Similar to lookup tables, linear models have low memory and processing requirements and they have been used for RL in current literature.

Artificial Neural Networks are quite popular in current literature for use as RL utility functions. This is not only because they enable generalization and abstraction but also because they can represent non-convex regions of the state space. This makes them incredibly useful when compared to lookup tables and linear models. ANNs also have small memory and processing requirements. A major drawback of ANNs is that they require significantly complex adjustments to the RL equations in order to be implemented as a utility function. Both linear models and ANNs are also somewhat of an art to model, requiring significant expertise to decide on such considerations as a suitable number of hidden layers or number of hidden nodes.

Cluster spaces are a form of RL utility function representation that allow for generalization and, thanks to high dimensional clustering equations in particular, also abstraction. This form of representation has been applied to RL equations in current literature and do not require as complex modifications to these equations as parametric approaches such as ANN. The major drawback to cluster based utility functions is that they have very high memory and processing requirements. This is because each data point used to train the representation needs to be stored in memory and then many of them need to be accessed when the model is in use. The longer the model is trained the bigger and slower it becomes. This renders cluster based representation as an unsuitable approach for our purposes.

Locality sensitive hashing is an approach that has not received any consideration for application to RL in the current literature that we are aware of. It has thus up to this point not been used as a RL utility function to the best of our knowledge. It does however have a lot of potential to be used for such a purpose. It enables both generalization and abstraction. As multiple hash maps need to be stored its memory requirements are slightly higher than that of lookup tables. Using a hash map has lower processing requirements than using a lookup table, so even though multiple hash maps are referenced at once the total processing requirements are still kept low. Because locality sensitive hashing has not been used as a RL utility function in current literature we cannot confirm how complex the required modifications to the RL equations would be. Based upon our experiences we would however assume that the required modifications would actually be quite low.

From Table 2 and the discussion that followed we can see that there are no obvious solutions to RQ2. Although locality based hashing provides both generalization and abstraction with reasonable memory requirements and low processing requirements it has not been used before as an RL utility function. This means that we cannot be sure of the extent to which the RL equations would need to be altered to enable its integration. In this thesis we present a representation method (Multi-Layer Hashing) that is based upon locality sensitive hashing. This method has all of the benefits of locality sensitive hashing presented in Table 2. We apply this method to RL as a utility function and find that the equational modifications required are really quite small. Thus the MLH method that we present in this thesis addresses RQ2 while filling a significant gap in the current literature.

2.2.5 Exploration

The assumption that we have made thus far is that an agent's sole factor in determining which action to select for execution is based upon the long term rewards that the agent will receive as a result of the selected action being executed. The actions taken do more however than influence the returned rewards according to the current utility function and model. They also contribute to learning the true utility function and model by affecting which state transitions and rewards are observed (Russell & Norvig, 2010). This aspect of action selection is vital for the agent's long term wellbeing. Action selection based solely on the maximization of long term rewards is referred to as exploitation, as the agent is exploiting its current knowledge to its own benefit. Action selection based solely upon the learning of the true utility function and model is referred to as exploration, as the agent is exploring different areas of the state and action space so as to improve its knowledge of the environment. An agent that focuses solely on exploitation may never have a full enough understanding of its environment to truly maximize its utility. Thus in focusing too much on its goal of utility maximization it ensures that it will never truly achieve it. An agent that focuses solely on exploration will maintain a true understanding of its environment but will never use this knowledge to maximize its utility, rendering the agent irrational and ultimately of little worth. An agent must therefore balance between exploitation and exploration activities if it is to truly prosper within its environment. There are various approaches to balancing between exploitation and exploration action selection. In this section we now discuss a number of these approaches.

2.2.5.1 Greedy

We mentioned earlier the extremity of focusing solely on exploitation action selection at the expense of exploration. An agent that takes this approach to action selection is referred to as a greedy agent (Russell & Norvig, 2010). A greedy agent will seldom truly maximize its utility. This is particularly true within deterministic, static environments as they do not vary from one time step to the next. In stochastic, dynamic environments greedy action selection, although still quite weak, fares a little better. This is because of the fact that the variations in returned rewards may alter the utility of the agent's actions from one time step to the next, which may have a knock on effect on which action is deemed the rational choice. For example, if action a_0 has the highest utility value from state s, then a greedy agent will always select it for execution. If however, due to stochasticity or a dynamic change in the environment the utility of action a_0 drops below the utility of any of its competing actions $a_{1\text{-}A}$ then the greedy agent will no longer select it, but will turn its attention to the action with the next highest utility value. As this process occurs over time the greedy agent unintentionally explores the actions that have what it thinks are the highest utility values.

2.2.5.2 ϵ-Greedy

Unlike greedy action selection the ϵ-Greedy approach explicitly performs exploration(Sutton & Barto, 1998). The simple mechanism that the ϵ-Greedy algorithm provides for this purpose is to select a random action with probability ϵ (a variable value ranging from 0 to 1) and to follow the greedy policy otherwise i.e. with probability $1 - \epsilon$.

The following diagram, taken from Sutton and Barto's pivotal book on RL (Sutton & Barto, 1998), illustrates the performance of the ϵ-Greedy algorithm with various values of ϵ when applied to a 10-armed bandit problem. This data used for the diagram was averaged over 2000 experimental iterations.

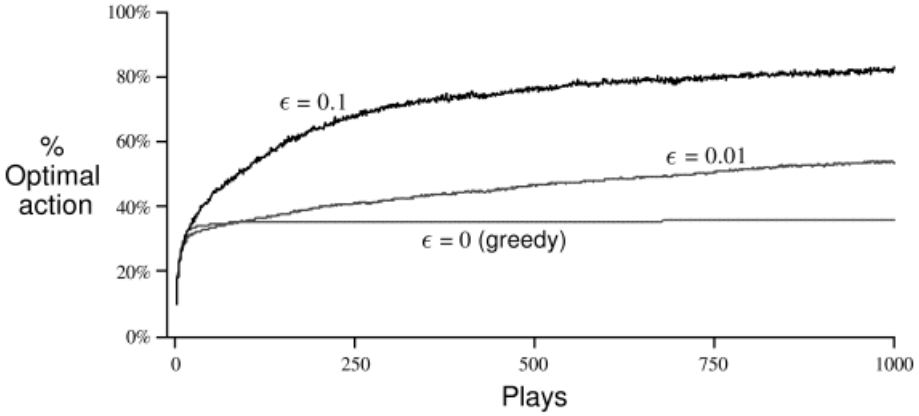

Figure 17 ϵ-Greedy performance, various values of ϵ in 10-armed bandit problem (Sutton & Barto, 1998)

The value of ϵ can also vary over time. This can enable an agent to perform more exploration immediately after initialization and then focus more on exploitation as time goes on. Besides of varying ϵ over time it is also possible to modify ϵ based upon some form of triggering event, such as error levels dropping below a threshold value, thus forcing the agent to reconsider its understanding of the environment. Although the ϵ-Greedy is a considerable improvement to the simple greedy approach it can still be quite slow. One reason for this is that it explores the environment unevenly; meaning that some areas in the state space will be explored more than necessary while others are not explored enough. A solution to this problem is to assign a separate ϵ value to each state, ensuring that exploitation is done when the agent finds itself in an area of the state space that it is quite familiar with already, while also exploring areas of the state space that it is not yet familiar with. This approach however is aimed at agents that use lookup tables or other similar approaches to represent the utility function and model. It does not carry over as well to agents using function approximation.

2.2.5.3 Boltzmann

Boltzmann a.k.a. softmax, action selection is another popular approach that balances exploration and exploitation throughout the life of the learning agent (Sutton & Barto, 1998). As opposed to ϵ-greedy action selection, in which all actions have an equal chance of being selected during an exploration step, Boltzmann action selection assigns a higher probability of selection *P(s,a)* to actions that have higher action-utility values. Boltzmann action selection is defined in equation (25).

$$P(s,a) = \frac{\exp \frac{Q(s,a)}{\tau}}{\sum_{i \in A(s)} \exp \frac{Q(s,i)}{\tau}} \tag{25}$$

where:

- τ is the temperature variable

A high temperature τ signifies that all actions are equally probable of being selected during an exploration step. This is equivalent to ϵ-Greedy action selection. A low temperature τ means that actions with higher action-utility values are more likely to be selected. This is equivalent to greedy action selection. As with ϵ-Greedy's ϵ variable the temperature τ variable value can be modified over time or as a result of some triggering event.

2.2.6 Coordination Graphs

The coordination graph framework was introduced to assist in large scale coordinated action selection (Guestrin et al., 2002). Within a coordination graph agents must determine a jointly optimal action without explicitly considering every possible action in their joint action space. Agents within a coordination graph do not coordinate their actions with all other agents within the system but rather with a subset of agents that have influence on or that are influenced by the agent. The set of state variables and actions that influence the agent are referred to as its scope. The agent thus does not maintain global joint state or action spaces but rather significantly smaller observable state space and relevant action space. In Guestrin's original presentation of the coordination graph framework it was assumed that each agent's scope is known in advance. This was later extended to enable the learning of interdependencies among cooperative agents (Kok, Jan't Hoen, Bakker, & Vlassis, 2005).

We will use the small four-agent system illustrated below as an example upon which we can base our explanation of action selection within a coordination graph.

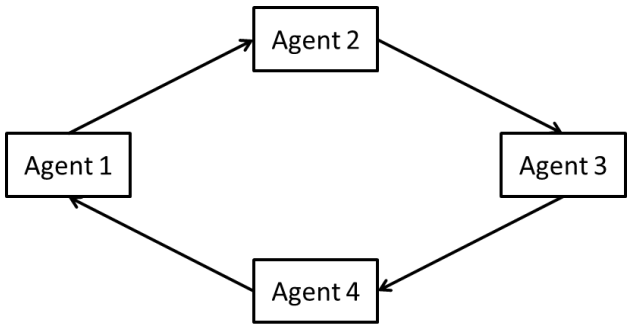

Figure 18 A four-agent coordination graph

The coordination graph framework divides a coordination game into a number of sub-games, each involving a subset of the total set of agents within the environment. Each subset consists of agents that are within each other's scopes. This approach assumes that the global reward function can be written as a linear combination of local reward functions, each involving a small number of agents. In the area of traffic control an intersection agent might for example only be concerned about the actions of its immediate neighbor agents while in robotic soccer an agent might only be concerned with a dynamic set of agents that are within its immediate vicinity. In our four-agent system, in which the action-utility function *Q* of each agent is only dependent on its own actions and that of its upstream neighbor we can write this as follows:

$$Q(a_1, a_2, a_3, a_4) = Q_1(a_1, a_4) + Q_2(a_2, a_1) + Q_3(a_3, a_2) + Q_4(a_4, a_3) \tag{26}$$

State *s* is held constant in this example and is thus omitted from the equations. Our objective here is to find the following:

$$\underset{a_1, a_2, a_3, a_4}{argmax}\ Q_1(a_1, a_4) + Q_2(a_2, a_1) + Q_3(a_3, a_2) + Q_4(a_4, a_3) \tag{27}$$

We can now apply Guestrin's variable elimination algorithm to optimize the agents action choices one at a time. We will begin by eliminating agent 1. We firstly group all functions that are dependent on agent 1's actions, getting:

$$\underset{a_2, a_3, a_4}{argmax}\ Q_3(a_3, a_2) + Q_4(a_4, a_3) + \underset{a_1}{argmax}\ [Q_1(a_1, a_4) + Q_2(a_2, a_1)] \tag{28}$$

For agent 1 to choose an optimal action it must know actions a_2 and a_4. Agent 1 must thus compute a conditional strategy e_1 as given below:

$$e_1(a_2, a_4) = \underset{a_1}{argmax} [Q_1(a_1, a_4) + Q_2(a_2, a_1)] \quad (29)$$

Agent 1 has now been eliminated from the equation, leaving us with the following:

$$\underset{a_2, a_3, a_4}{argmax} Q_3(a_3, a_2) + Q_4(a_4, a_3) + e_1(a_2, a_4) \quad (30)$$

We now move on to eliminate agent 2 in much the same way as we eliminated agent 1.

$$\underset{a_3, a_4}{argmax} Q_4(a_4, a_3) + e_2(a_3, a_4) \quad (31)$$

We now eliminate agent 3 following the same process:

$$e_4 = \underset{a_4}{argmax} e_3(a_4) \quad (32)$$

Agent 4 is now free to choose an action a_4^* that maximizes e_3. Agents 3, 2, and 1 are now obliged to perform the actions that they committed to that lead to the maximization of a_4^*. The process so far however has only given the optimal action a_4^* while all other actions are only conditional actions. In order to compute the optimal actions for the other agents we must run the algorithm again with a different agent being the last to be eliminated and with plugging action a_4^* into the equation as agent 4's conditional action. Following this procedure all optimal actions can be obtained, regardless of the initial order that the agents were eliminated in. The execution time however increases exponentially with the increasing width of the coordination graph. This challenge has been noted and efforts have been made to address it, for example the max-plus algorithm for coordination graphs (Kok & Vlassis, 2006).

2.3 Traffic Control

In this section we introduce the author to the traffic engineering concepts and terminology relevant to this thesis. We then describe classical approaches to traffic control that are commonly implemented throughout the world. We then describe a selection of AI based traffic control systems with a focus on RL agent based systems.

2.3.1 Traffic Engineering Concepts and Terminology

A **transport network** consists of a potentially very large number of uncontrolled and signalized intersections that are interlinked by means of roadways. Whereas an **uncontrolled intersection** is managed by traffic rules a **signalized intersection** is managed by a **traffic signal controller**. This signal controller is a combination of software and hardware that is usually contained within a metal box in close proximity to the intersection. This signal controller implements a set of **signal timing plans** that dictate the order and timing of the intersection's traffic lights. These lights inform vehicles on all of the intersection's approaches whether or not they have the right to pass through the intersection. For the interval of time that a traffic light turns green for any given approach the vehicles on that approach have right of way to proceed through the intersection. This is known as a **green interval**. A **red interval** denies vehicles on an approach access to the intersection. A **change interval**, indicated by an amber traffic light, indicates a change from a green interval to a red interval (In this thesis we assume Irish rules of the road and traffic light sequencing in which vehicles are driven on the left of the road and change intervals follow green intervals and not red ones). During a change interval vehicles that are approaching the intersection should stop if safe to do so. This interval generally lasts for about three seconds in practice. A **clearance or all-red interval** occurs when vehicles on all approaches of the intersection are prohibited from proceeding. This is a short interval, typically lasting one or two seconds, that occurs for safety purposes just after any change interval. A **phase** consists of a combined set of non-conflicting green intervals followed by a change interval and a clearance interval. During this time any intersection approaches that do not have a green interval experience a red interval. The duration of a phase is referred to as the **phase length**. Phases occur in sequence and one full rotation of a sequence of phases is referred to as a **cycle**. The duration of one complete cycle is called the **cycle time**. The **offset** between two intersections is defined as the time difference between the start of their respective cycles. Signal timing plans, also known as **phase plans**, can be illustrated using **phase diagrams** or using **ring diagrams**. One possible phase plan for the intersection illustrated in Figure 19 is represented using the phase diagram in Figure 20 and also the ring diagram in Figure 21.

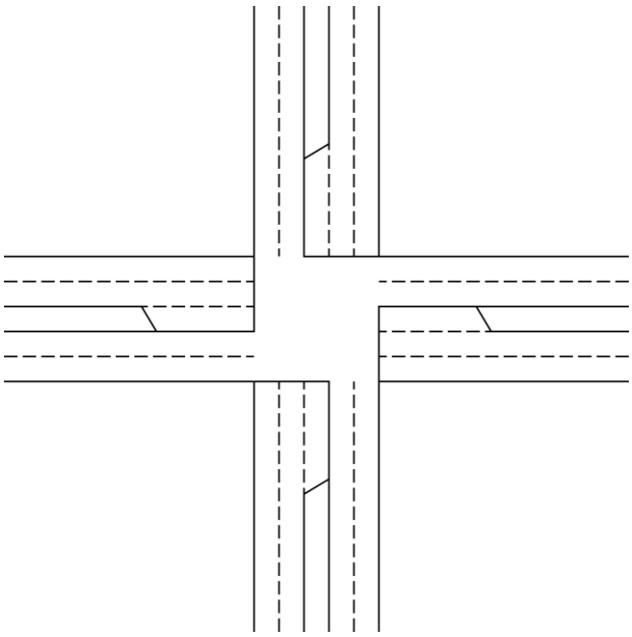

Figure 19 Intersection layout

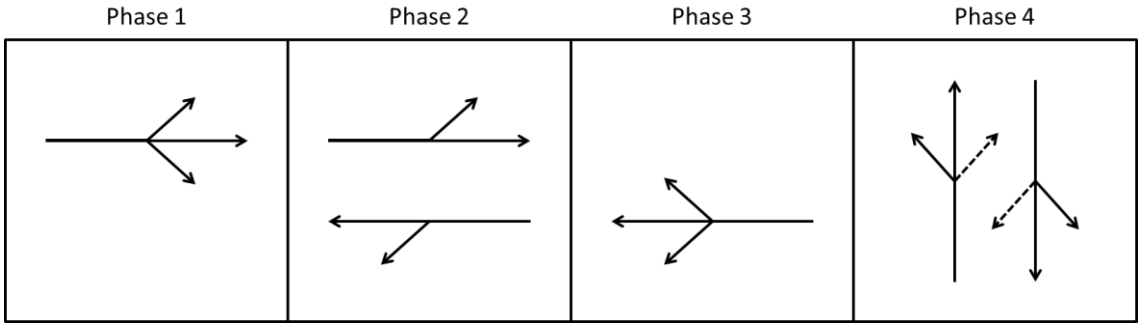

Figure 20 Phase diagram

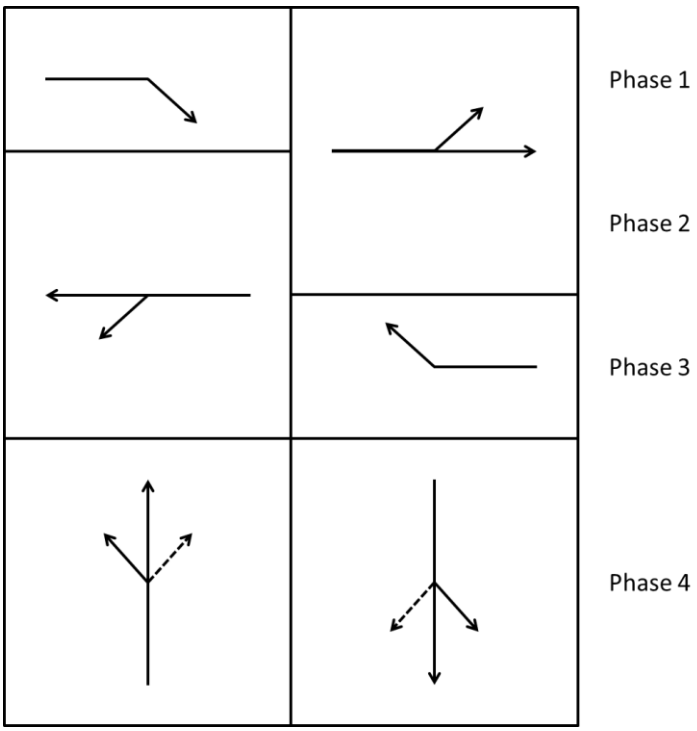

Figure 21 Ring diagram

When cycle times, phase lengths, and offsets of signal controllers along a **traffic corridor** or **arterial** are set correctly a **progressive signal system** can be established that creates "green waves" of traffic. An essential element of establishing and maintaining progressive signal systems is for each intersection along the traffic corridor to have the same cycle length, or multiples of the same cycle length. The time difference between the first and last vehicles that pass through the arterial without stopping is called the **bandwidth**. The **time space diagram** presented in Figure 22 illustrates a progressive signal system.

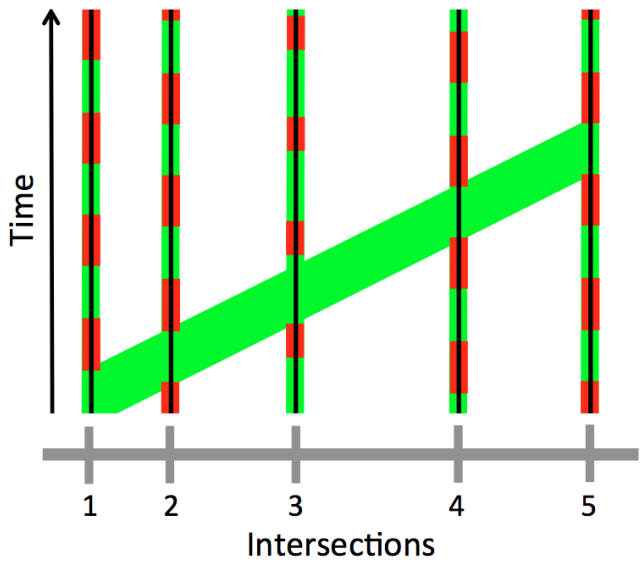

Figure 22 A progressive signal system

Where there is little traffic congestion, traffic controllers along a main arterial will give a green signal just before a platoon of vehicles arrives. This is referred to as **forward progression**. This is the type of progression established in Figure 22 along the traffic corridor from intersections 1-5. In congested conditions the traffic corridor's upstream traffic controllers begin their green phases first so as to clear congested traffic before letting any more vehicles into the arterial. This is referred to as **reverse progression**. When intersections are located at a very short distance from each other it may be more appropriate to have all traffic controllers begin their green intervals at the same time. This is referred to as **simultaneous progression** and is the easiest form of progression to achieve. **Flexible progression** refers to when different forms of signal progression are activated at different times of the day or under various traffic conditions.

Establishment and maintenance of progressive signal systems is much more complex in transport networks where traffic flow is dynamic. Dynamics can lead to the direction of the main flow of traffic along a traffic corridor to reverse within a matter of hours. It can also lead to changes in the course of a traffic corridor. These course changes can be temporary, such as those that might be caused by road closures. They can also be permanent, such as those that might be caused by permanent alterations to the transport network topology.

Establishing progressive signal systems also becomes much more complex in two directional traffic corridors. In these scenarios progression in one direction may decrease the amount of possible progression in the opposite direction. This can be seen in Figure 23, where forward progression has been established along the traffic corridor running from intersections 1-5 and no such progression has been established along the same traffic corridor running from intersections 5-1.

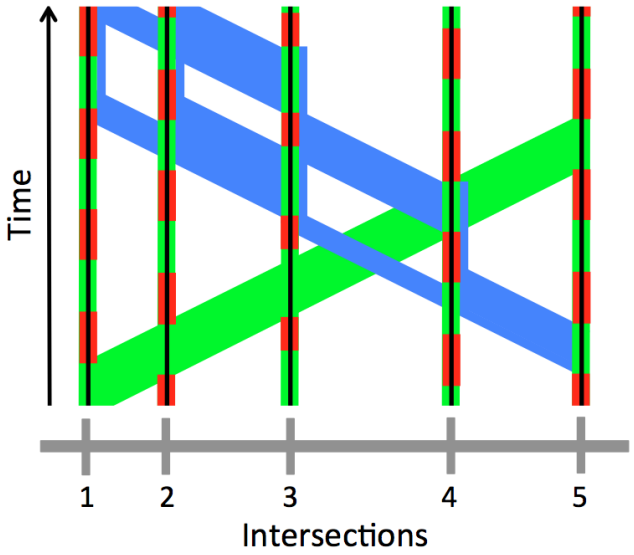

Figure 23 Challenge with progression in a two directional traffic corridor

Traditional solutions to this problem include designing the transport network with suitable distances between intersections to enable signal progression in both directions, as seen in Figure 24, or establishing progression in the direction that has a significantly higher flow of traffic.

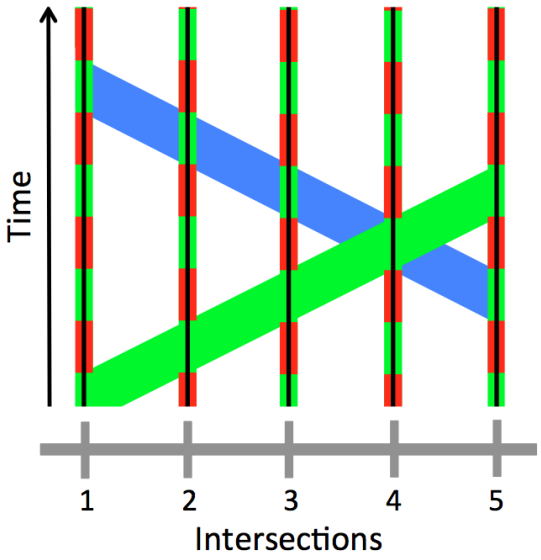

Figure 24 Two-way progression

There are three main levels of **scope** in which a traffic controller may act. The most basic and easiest to achieve of these levels of scope is **isolated scope**, in which the traffic controller functions independently of any other traffic controllers. Within this scope no consideration is given to offset,

coordination, or progressive signal systems. Actuated traffic controllers (see section 2.3.2.2) generally function within this scope. The next level of scope that a traffic controller may act in is that of **arterial scope**. Within this scope traffic controllers generally coordinate with upstream and downstream traffic controllers along a traffic corridor. The third and final scope in which a traffic controller may act is that of **grid scope**. To coordinate within a grid network each intersection increases its scope so as to take into account upstream and downstream traffic controllers in multiple directions. Traditionally coordination within a grid network is dealt with by logically dividing the network into non-overlapping arterial segments in which no loops exist (Roess, Prassas, & McShane, 2011, p 695). This process requires time, effort, and expert knowledge but prevents the problem from becoming excessively computationally expensive. Essentially this approach extends the idea of coordination within an arterial scope. Optimization software such as TRANSYT (Roess, Prassas, & McShane, 2011, p 718) and PASSER (Roess, Prassas, & McShane, 2011, p 718) are typically used to aid traffic engineers through this process. Another approach to coordination within a grid scope would be to design the physical grid network with suitable distances between intersections to enable signal progression.

Traffic signal controllers are often connected to **sensors** that are embedded within the traffic network. These sensors enable the signal controller to detect vehicles as they approach the intersection. Different types of traffic detectors include ultrasonic detectors, magnetic detectors, microloop detectors, cameras, and **inductive loop detectors**. Inductive loop detectors are the most popular of these as they are relatively cheap yet quite reliable. An inductive loop is installed by laying an electrically conductive cable into a shallow saw cut in the road and then refilling the cut with an epoxy sealant. The saw cut tends to be a square, rectangle, or trapezoid. The loop is connected to a low-grade electrical source, which creates an electromagnet that can detect whenever a metallic object e.g. a vehicle, moves over it. Figure 25 shows a typical induction loop detector installation.

Figure 25 Typical induction loop detector installation

2.3.2 Classical Approaches to Traffic Control

Traffic signal controller operation is usually divided into the following categories: pre-timed, actuated, and adaptive. These approaches differ to each other with regards to complexity, flexibility in their phase plans, as well as with regards to the amount of information observed from sensors and how this information is used. We will now discuss each of these approaches to traffic control in turn.

2.3.2.1 Pre-timed

Pre-timed traffic control (Roess, Prassas, & McShane, 2011) is the most basic traffic control method and is the most common in practice due to the fact that it does not require any sensors or means of communication between traffic signals. It is thus relatively cheap to install and is particularly dependable in situations where traffic patterns do not change significantly over time. This method fixes the cycle lengths and phase lengths, and subsequently offsets, over set periods of time. These fixed timing plans can be optimized offline based on observed or historical traffic flow volumes. This optimization can be achieved using algorithms such as Webster's algorithm (Webster, 1958) or using software such as TRANSYT (Robertson, 1969). Signals can be optimized for different levels of traffic flow and then at particular times of the day the signal plans can be changed. Thus pre-timed traffic control can handle predictable changes in traffic flow such as weekday peak traffic times. Pre-timed control cannot however dynamically adapt to changes in traffic flow and must be regularly reprogrammed to handle evolving traffic patterns. This can be time consuming for a traffic engineer. Multiple pre-timed traffic controllers can be programmed to work in coordination so as to establish a progressive signal system. This is achieved by setting appropriate offset times between adjacent intersections.

2.3.2.2 Actuated

Actuated traffic control (Roess, Prassas, & McShane, 2011) is achieved when traffic controllers set phase lengths based upon sensor vehicle detections known as actuations, as opposed to setting pre-timed phase plans. Although the signal controller still cycles through a specified sequence of phases the phase durations are not fixed. A phase can even be skipped if no traffic is detected on its associated approaches. Initially, a minimum green time is allocated to each phase. When a vehicle is detected on one of the phase's approaches nearing the end of the phase duration then the phase duration can be incremented by a specified extension time. When a vehicle is detected on an approach assigned to another phase then a limit is set on the current green time. Through this process phases are given green time in proportion to their current traffic flows. An actuated phase is illustrated in Figure 26.

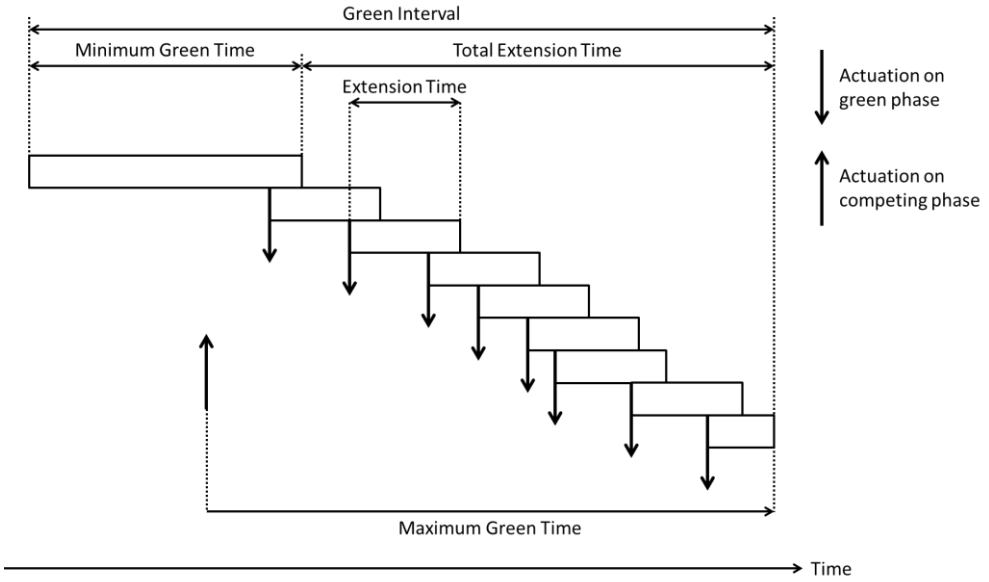

Figure 26 Actuated phase

During heavy traffic conditions an actuated traffic controller will essentially reduce to a pretimed traffic controller that assigns the maximum green time to each phase. Actuated traffic control does not require any form of communications between traffic controllers, which reduces costs but also means that no coordination is possible. This form of traffic control is thus mainly suitable for more isolated intersections that have low to medium traffic flow levels.

2.3.2.3 Adaptive

Adaptive traffic control (Roess, Prassas, & McShane, 2011) is the most complex and expensive of the three forms of traffic control and can require multiple sensors at each intersection as well as communications between intersections. Forms of communication between intersections range from cables such as ADSL or fiber optic cables to wireless communications such as Wi-Fi. Adaptive controllers use the data that they detect from their local sensors as well as data communicated to them from other traffic controllers to adapt to changes in traffic flow. This gives this approach the potential to adaptively optimize traffic flow across multiple intersections. Varied approaches have been proposed that strive to achieve efficient adaptive traffic control. In this section we will review some of the more mature and widely deployed approaches before looking into more advanced and experimental approaches.

2.3.2.3.1 Sydney Coordinated Adaptive Traffic System (SCATS)

SCATS (Sims & Dobinson, 1980) is an adaptive traffic control system that was developed in the early 1980s in Sydney, Australia by the New South Wales roads and traffic authority. It is currently deployed

in 27 countries worldwide controlling more than 37,000 intersections (www.scats.com.au, accessed on 06/04/2015). In traffic networks in which SCATS is to be implemented a traffic engineer will divide the network up into multiple regions. Each local traffic controller within a region receives instructions from a regional master controller. Each regional master controller within the system reports to a control center, which is mainly used for administration purposes. SCATS thus has a hierarchical architecture that is illustrated in Figure 27.

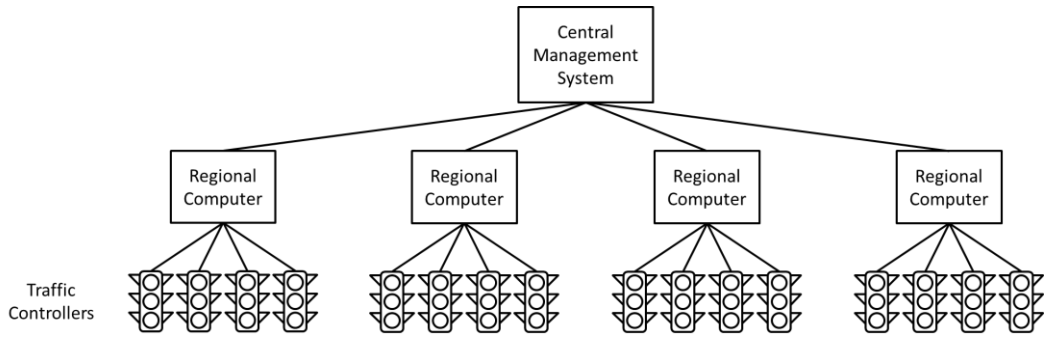

Figure 27 SCATS architecture

Traffic engineers must predefine the traffic signal timing plans for each region within the network. These plans give the best phase durations and offsets for local controllers within the region based on specific traffic flow conditions. Traffic sensors detect the regional traffic flow conditions and the appropriate plans are employed. Thus SCATS can establish coordination within a region and can also adapt to changing traffic conditions. The process of designing the timing plans for each region in the traffic network can be quite tedious and prone to error.

2.3.2.3.2 Split, Cycle and Offset Optimization Technique (SCOOT)

SCOOT (Hunt, 1981) is an adaptive traffic control system that was developed in the early 1980s in the UK at the British Transport and Road Research Laboratory (TRRL). SCOOT is implemented in over 200 cities throughout the world, but most predominantly in the UK (www.scoot-utc.com, accessed on 06/04/2015). SCOOT has been shown to reduce traffic delay by approximately 11% under peak conditions and 16% under off peak conditions (Klein, 2001). It has been shown though that SCOOT's performance begins to degrade under saturated traffic conditions (Papageorgiou, Kiakaki, Dinopoulou, Kotsialos, & Yibing Wang, 2003). SCOOT can be looked on as an online version of TRANSYT (Klein, 2001), which as we mentioned in section 2.3.2.1 is used to optimize pre-timed timing plans (Robertson, 1969). SCOOT has a centralized architecture, where a central computer receives remote traffic sensor data, optimizes the signal timings, and then sends any changes out to the local signal controllers. SCOOT minimizes the average sum of vehicle queues in an area as well as the number of times vehicles need to stop. Minimizing the number of times vehicles need to stop is done by optimizing the bandwidth within

green waves. SCOOT's optimization routine consists of a split optimizer, offset optimizer, and cycle optimizer, which each decide once every five minutes whether or not to modify the split (ratio of each phase length to the cycle length), offset, or cycle length respectively by a few seconds. Similar to SCATS, transport networks that implement SCOOT need to be divided up into regions by a traffic engineer. Each traffic controller within a region maintains a common cycle length. Coordination can be established within each region. Unlike SCATS however SCOOT does not merely implement different predefined plans in response to changing traffic conditions.

2.3.3 Artificial Intelligence Based Traffic Control

The literature regarding AI based approaches to traffic control is extensive and quite diverse. It is beyond the scope of this thesis to review such a vast range of literature. The purpose of this subsection is thus to review a representative selection of AI based traffic control systems that are relevant to this research. These approaches are varied in architecture ranging from centralized to single-agent or multi-agent control. They are also varied in the technologies that they employ. Examples of different technologies employed in AI based traffic control systems include fuzzy logic (L. Zhang, Li, & Prevedouros, 2012), neural networks (Srinivasan et al., 2006), ant colony optimization (Putha & Quadrifoglio, 2010), swarm intelligence (de Oliveira, Ferreira, Bazzan, & Klügl, 2004), constraint optimization (de Oliveira, Bazzan, & Lesser, 2005), genetic algorithms (Girianna & Benekohal, 2004), organic computing (Prothmann et al., 2008), dynamic programming (Porche & Lafortune, 1997), Q-Learning (Abdulhai, Pringle, & Karakoulas, 2003). We then review traffic control approaches that specifically focus on MDP based learning techniques such as the ones that we describe in sections 2.2 and i.e. DP, SARSA, Q-Learning, and ACRL.

2.3.3.1 Centralized

Centralized traffic control systems consider the entire transport network as a single complex system. Any computation or memory requirements are the responsibility of a central processing device. This contrasts with the distributed agent based approach that we focus on in this thesis. Centralized systems tend to suffer from state space explosion as well as other issues such as bottlenecking, over dependency on communications, scalability issues, etc.

Girianna and Benekohal (Girianna & Benekohal, 2004) have proposed a centralized traffic control method that can establish coordination between intersections within an oversaturated transport network. This algorithm specifically coordinates signal timing plans along traffic corridors that overlap each other within a grid network. Traffic corridors that carry equal levels of traffic are assigned an equal priority while saturated traffic corridors are given higher priority than those crossed along their paths. The proposed algorithm models the network signal coordination problem as a large combinatorial

optimization problem. It then uses a micro Genetic Algorithm (μGA) to solve the problem. The evaluation of this system was carried out using 20 simulated intersections, but to reduce scalability issues and complexity a number of limitations were put in place. Such limitations include: each intersection only had two phases, no turning movements were permitted at any intersection, all arterials were one-way streets, all arterials were considered to be oversaturated i.e. no adaptation to changing traffic conditions, and coordinated corridors had to be defined by the user (predetermined coordination within an arterial scope). As well as these limitations the algorithm could only handle one single arterial that crosses multiple parallel arterials that run perpendicular to it. Thus loops are never present in the network. Despite these limitations this study is very useful as it is one of the few attempts at coordination of multiple corridors within a grid network.

2.3.3.2 Independent Agents

As opposed to the centralized approach described in the previous section the agent based approach distributes the processing and memory requirements of the traffic control system among a dispersed set of software agents. This approach is scalable, robust, and flexible. A common approach to agent based traffic control systems is to model each intersection as a single agent that assumes complete independence from any other intersection within the transportation network. This is a fully distributed approach to traffic control. This has the benefits of having significantly reduced state space size and comparatively low computation and memory requirements. It also has no need of communications between agents within the system. It does however ignore the complex interdependencies of agents within the system.

Bull et al. (Bull et al., 2004) propose an independent agent traffic control that uses a Learning Classifier System (LCS). Using this approach each intersection within the transport network is modeled as an independent agent. The agent's goal is to find the optimal phase lengths given a predefined set of traffic flow levels. The algorithm was tested on an intersection with four approaches and only two phases i.e. a north-south phase and a west-east phase as illustrated below:

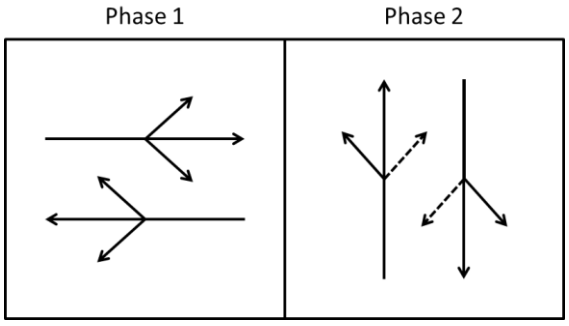

Figure 28 Two phase plan

The algorithm was then tested with various combinations of state variables, LCS rewards, and actions. It was also tested with a more detailed intersection layout. It was found that in some cases and with some setting specifications the LCS approach outperformed standard control, indicating the approaches potential application to traffic control.

An early example of applying fuzzy logic to traffic control is given by Pappis and Mumdani (Pappis & Mumdani, 1977). Similar to the example above a simple two phase intersection was used for testing. In this approach however an actuated traffic control technique was used in which the agent must decide every ten seconds whether or not to extend the current phase's length depending on current traffic volumes on all of the intersection's approaches. Sazi Murat et al. (Murat & Gedizlioglu, 2005) later extend this idea to more complex intersections (more approaches, more lanes, more phases, and two-way traffic) and also enable the agent to modify the phase sequences. Results showed an improved performance of between 15% and 50% when compared to traditional actuated traffic control.

2.3.3.3 Coordinated Agents

As opposed to the independent approach to agent based traffic control described in the previous subsection this section describes approaches in which agents coordinate their traffic control efforts. This is an approach whose agents recognize that there are complex interdependencies between themselves and other agents within the system.

Organic Computing was suggested as the base of a coordinated agent traffic control system proposed by Tomforde et al. (Tomforde et al., 2008). This decentralized approach adaptively coordinates traffic along a traffic corridor. Prioritized traffic corridors are not user defined but are discovered automatically based on the traffic flow levels travelling along them. It is assumed that all controllers within a coordinated traffic corridor have the same cycle length and that distances and average speeds between intersections are known. The traffic corridors are optimized in one direction, though this direction or even the arterials path may change, if the weight of traffic flow shifts. As an Organic Computing system this proposed system uses a microscopic traffic simulator to optimize traffic light controller parameters off-line. Evolutionary Algorithms (EA) are used in optimizing these parameters. Optimized parameters can then be selected in real time. A drawback to this approach is that the system needs to accurately model the transport network in a microscopic traffic simulator. This leads to significant requirements of time, expertise, processing power, and memory.

Srinivasan et al. (Srinivasan et al., 2006) present a hierarchical agent based approach to traffic control in which a number of AI techniques, particularly Artificial Neural Networks (ANN), are used to optimize the flow of traffic within a transport network. Of particular note in this research is the complexity of the evaluation in terms of the transport network size and detail and also the fluctuating traffic flow levels.

The evaluation was done using the PARAMICS industry standard microscopic simulator. Results showed that this algorithm had an impressive reduced delay of up to 78% when compared to the SCATS system.

Kosonen (Kosonen, 2003) introduces a multi-agent approach to traffic control using fuzzy inference techniques. Kosonen's approach to traffic control is to have each of an intersection's phases modeled as a separate agent. Agents coordinate to decide the duration of phase lengths as well as which phase has rights to the green interval. Decisions are made based upon traffic volumes on the intersection's approaches. Thus intra-intersection coordination is more of a focus than the more common approach to inter-intersection coordination. Although it is mentioned in Kosonen's work that inter-intersection coordination is possible the details of such additional functionality are not described.

The final example that we will look at (Bazzan, 2005) is a slight exception to agent coordination in that the agents themselves are in fact independent agents that do not explicitly communicate with each other to share information in any way. Coordination is however established within the transport network given one particular assumption, that the reward that each agent is individually optimizing is in fact a single shared global reward. This approach is thus a coordination game being played among the intersection agents. Evolutionary game theory principles are used by the agents in selecting from among a predefined set of phase plans. This approach is demonstrated in a scenario of a ten intersection arterial. Each agent must choose which direction through its intersection is to be favored. Coordination is then implicitly established when all intersections along the arterial optimize the same direction.

2.3.3.4 Reinforcement Learning Based Approaches

This section reviews approaches to traffic control that specifically focus on RL based techniques such as the ones that are described in sections 2.2 i.e. DP, SARSA, Q-Learning, and ACRL.

2.3.3.4.1 Centralized Learning

Centralized approaches to learning in a traffic control system tend to be highly dimensional as the number of state variables increases exponentially with each additional intersection. Due to this fact some form of function approximation would need to be used to represent the composite utility function. Such is the case with the centralized traffic control system proposed by Prashanth and Bhatnagar (Prashanth & Bhatnagar, 2011). In this approach all intersections within the transport network are modeled using a single MDP. This approach was evaluated on a grid transport network of 333 intersections.

2.3.3.4.2 Independent Learning Agents

Independent traffic control learning agents can be considered single-agents and as such can potentially use any of the single-agent learning algorithms that are described in section 2.2.

The Adaptive Limited Lookahead Optimization of Network Signals – Decentralized (ALLONS-D) (Porche & Lafortune, 1997) is a DP approach to traffic control. ALLONS-D uses vehicle arrival

information from upstream sensors when choosing which phase should be assigned the current green interval. ALLONS-D optimizes for minimum delay. ALLONS-D agents do not explicitly coordinate with each other however implicit coordination is assumed due to the fact that the data being used is taken from upstream sensors. A hierarchical version of ALLONS-D does exist that does explicitly establish coordination.

An early application of RL to traffic control is that of Thorpe's SARSA based approach (Thorpe, 1997). In this approach the signal controller can decide whether to let pass either the north bound vehicles or south bound vehicles on a four approach intersection. The decision is made based upon queue lengths and the time since the previous phase change. The SARSA agent controlled intersection thus aims to reduce queue lengths as much as possible. An interesting aspect of this research is the differentiation between learning difficulty of agents at intersections in different areas within the grid network. For example it was shown that learning results were better on intersections in the center of a 4x4 grid than those on edge intersections, which were in turn better than those on corner intersections.

Q-Learning has proven to be a very popular approach to independent learning agent traffic controllers. Camponogara and Kraus (Camponogara & Kraus, 2003) take such an approach. In this system queue lengths are used as state variables within the state space and the agent can perform phase length modification actions. The reward used is inversely proportional to the queue length. It was found that Q-Learning lead to higher improvements in waiting time when neighboring intersections also implemented Q-Learning, thus illustrating implicit learning.

Abdulhai et al. (Abdulhai et al., 2003) also implemented independent agent traffic control using Q-Learning. They found that this approach performs as well or better than pre-timed control under constant traffic conditions but also that it outperforms pre-timed control by 38% - 44% in fluctuating traffic flow conditions. As opposed to pre-timed control's cyclic approach this approach is more of an actuation based approach, which is naturally more adept at handling fluctuating traffic conditions. In this approach the agent must choose whether to continue with the current phase or change to the next one. Queue lengths as well as elapsed phase times are used as state variables within the state space. The reward is coded as the inverse of the average delay.

Weiring (Wiering, 2000) presents a model-based RL approach to traffic control within a small grid network. The traffic controller agents goal is to minimize vehicle waiting times within the transportation network. An interesting aspect of this research is that although the intersections are modeled as the agents within the MAS the state variables within the state space are based upon the waiting times of individual vehicles.

Neuro-Fuzzy Actor-Critic Reinforcement Learning (NFACRL) (Jouffe, 1996) was proposed for use in a traffic control system in a study conducted by Zhang et al. (Y. Zhang, Xie, & Ye, 2007). As discussed in section 2.2.2.3.3 ACRL has the advantage of being able to use various different technologies to implement the utility function, model, and even policy. This method takes advantage of this by

implementing the value function as a neuro-fuzzy network. This was to assist in handling large state spaces. Zhang et al. implemented two NFACRL based traffic control systems. One used a fixed phase sequence, while the other used a variable phase sequence. Both approaches were applied to both an isolated intersection and to an arterial of intersections. In both cases the agents used were independent agents that did not explicitly communicate with each other. Evaluation of these systems found that the system with phase sequence selection constantly outperformed the system with fixed phase sequences in isolated intersections. Unsurprisingly, the system with variable phase sequences did not perform well along the arterial of intersections. This is unsuprising as cycle lengths were not consistant across all intersections along the arterial and thus coordination is impossible.

2.3.3.4.3 Coordinated Learning Agents

This subsection reviews a selection of literature regarding coordinated learning agent based traffic control that is most similar to the vein of research investigated in this thesis. These approaches to traffic control model intersections as learning agents that coordinate with each other during their action selection and learning processes.

Salkham et al. (Salkham & Cahill, 2010) use a technique named Collaborative Reinforcement Learning (CRL) as a basis for a traffic control system named Adaptive Round Robin (ARR-CRL). CRL is a form of Q-Learning that allows collaboration between CRL agents. CRL Agents collaborate by exchanging remote policies with their neighbours. These policies are then taken into account, as well as an agent's local policy, when making decisions as to what actions should be taken next. ARR rewards are based upong queue length and vehicular throughput while intersection agent actions allow for setting phase time durations as well as the ability to skip a phase entirely. Agents periodically exchange reward information with their immediately adjacent neighbors and incorporate this information into their own reward. As an agent's neighbors reward has an influence on the agent's actual reward then the agent is encouraged to perform actions that are good for both. ARR was evaluated on a large scale simulation (60+ intersections) using a custom microscopic simulator and performed well when compared to the performance of Round Robin control and the SCATS based algorithm named SAT.

W-learning (see section 2.2.3.2.2.1) was applied to traffic control in a doctoral thesis presented by Dusparic (Dusparic, 2010). This approach is thus a multi-policy multi-agent form of learning for traffic control. Results showed an improvement in average vehicle waiting times of up to 73% and 88% when compared against Round Robin and SAT algorithm. Results also showed that if policies are conflicting the performance of higher priority policies will be improved with only small negative effects on the performance of lower priority policies.

Bazzan et al. (Bazzan, de Oliveira, & da Silva, 2010) propose a team learning based approach to traffic control. Using this method agents are organized into teams of limited size so as to reduce their action space sizes. Each agent learns its own optimal policy but within each group a supervisor agent

maintains a broader view of the actions and their effects within the team as a whole. Thus coordination is established within the teams by the supervisor agents making suggestions of alternate actions that can lead to more global reward optimization. This approach is however vulnerable to a single point of failure within each team. If a supervisor agent does fail then the agents within the group revert to independent Q-Learning.

Richter et al. (Richter, Aberdeen, & Yu, 2007) propose a traffic control system based on Natural Actor-Critic (NAC) RL (Peters, Vijayakumar, & Schaal, 2005). This approach optimizes vehicle throughput at each intersection within a grid network composed of four phase signal controllers. Agents communicate with their immediate neighbors in order to establish coordination. Evaluation of this method was done on a number of different traffic patterns and on a grid network of up to 10x10 intersections. It was found that the algorithm could outperfom the SAT algorithm but needed to learn for approximately three days of real world time before being able to do this, as an intersection only begins to learn usefull information after its downstream neighbours have already converged on optimal policies.

Kuyer et al. (Kuyer, Whiteson, Bakker, & Vlassis, 2008) propose a coordinated traffic control system based upon coordination graphs (see section 2.2.6 for details on coordination graphs). This approach focuses on neighboring signal controller agents collaboratively agreeing on which actions should be taken. This approach assumes that dependencies exist only between direct neighbor agents as the algorithm would suffer from state and action space explosion otherwise. Kuyer's approach uses the max-plus algorithm in solving the coordination graph as opposed to the variable elimination algorithm (see section 2.2.6) so as to reduce the impact of higher levels of network scale.

2.3.4 Conclusion

In this section we have described a wide variety of traffic control methods. These methods range from classical, to AI based, to specifically RL based. They vary in their architectures, their abilities to establish progressive signal systems, their abilities to learn, etc. The first of the two research questions that we address in this thesis (RQ1) focuses on the detection of a method of intersection coordination that can automatically create dynamic progressive signal systems within a multi-agent transport network using learning techniques, Reinforcement Learning techniques in particular. In this section the question remains to be answered as to the suitability of each of the traffic control methods presented as adequate solutions to RQ1. In order to assess the suitability of each method we have composed the following table, which we will now discuss in more detail.

Traffic Control Method	Coordination	Establishes Progressive Signal System	Architecture	Adaptive	Learning	Setup
Pre-Timed	No	Yes	Distributed	No	None	Complex
Actuated	No	No	Distributed	Yes	None	Easy
SCATS	Yes	Yes	Hierarchical	Yes	None	Complex
SCOOT	Yes	Yes	Centralized	Yes	None	Easy
Girianna	Yes	Yes	Centralized	No	Genetic	Complex
Bull	No	No	Distributed	Yes	LCS	Easy
Pappis	No	No	Distributed	Yes	Fuzzy Logic	Easy
Organic	Yes	Yes	Distributed	Yes	Organic	Complex
Srinivasan	Yes	Yes	Hierarchical	Yes	ANN	Complex
Kosonen	Yes	No	Distributed	Yes	Fuzzy Logic	Easy
Bazzan (2005)	No	Yes	Distributed	Yes	Evolutionary Game	Complex
Prashanth	No	No	Centralized	Yes	RL	Easy
Allons-D	No	No	Distributed	Yes	DP	Easy
Thorpe	No	No	Distributed	Yes	SARSA	Easy
Camponogara	No	No	Distributed	Yes	Q-Learning	Easy
Abdulhai	No	No	Distributed	Yes	Q-Learning	Easy
Weiring	No	No	Distributed	Yes	MBRL	Easy
Zhang	No	No	Distributed	Yes	NFACRL	Easy
Salkham	Yes	No	Distributed	Yes	CRL	Easy
Dusparic	Yes	No	Distributed	Yes	CRL	Easy
Bazzan (2010)	Yes	No	Hierarchical	Yes	Stochastic Game	Complex
Richter	Yes	No	Distributed	Yes	NACRL	Easy
Kuyer	Yes	No	Distributed	Yes	Coordination Graph	Easy

Table 3. Traffic control method comparison

In Table 3 each traffic control method that we presented in this section is represented by a row, in whose first cell is either the method's name, or in the cases of control methods' whose names are unclear, the name of the first author of the academic paper describing the method. Table 3 also presents the following information for each traffic control method discussed.

- The control method's ability to coordinate actions between intersection controller agents
- Whether it enable progressive signal systems to be established within the transport network
- The control method architecture types

- Whether the method can dynamically adapt to changing traffic flow conditions
- The learning method being implemented
- How complex it is to set up the traffic controllers. For example, if an expert traffic engineer is needed to come out and perform analysis on an intersections traffic flow patterns over a matter of days in order to set up or update a traffic controller, as is the case with pre-timed control and SCATS, then this would be looked on as complex.

With regards to RQ1 the most important categories of Table 3 are "Establishes Progressive Signal Systems" and "Learning" as these directly address the question. The other categories have been added however as they are also important to be taken into consideration. For example, we must consider carefully if we would really consider implementing a centralized approach that is very complex to set up.

The first traffic control method presented in Table 3 is that of pre-timed control (Roess, Prassas, & McShane, 2011). This method is one of the most commonly implemented methods of traffic control throughout the world. One major reason for this is that it does not require expensive sensors in order to function. This however leads to it being unable to dynamically adapt to changes in traffic flow levels. Although traffic controllers using this method do not have any means of communicating and coordinating their actions with one another they are able to establish progressive signal systems throughout the network. This ability to establish progressive signal systems is made possible by expert traffic engineers who analyze the network's traffic flow levels and patterns and use the information gained to carefully design phase sequences that enable such systems. Thus pre-timed control is complex to set up. Due to its complexity in setting up, its inability to learn or to even adapt to dynamic changes in traffic flow patterns it is not a suitable solution to RQ1.

The next traffic control method discussed is that of actuated control (Roess, Prassas, & McShane, 2011). This method is quite popular in practice due to the fact that it is relatively simple to set up and once set up the actuated traffic controllers are able to adapt to dynamic changes in traffic flow levels and patterns. Adaptive controllers do not coordinate their actions with each other. This means that they are quite limited in that they do not enable the creation of progressive signal systems. For this reason, and for the fact that actuated control does not have the ability to learn, we deem it an unfit solution to RQ1.

SCATS (Sims & Dobinson, 1980) and SCOOT (Hunt, 1981) are both traffic control methods that have been popular adaptive methods of traffic control since the early 1980s. This is because they enable the establishment of progressive signal systems as well as enabling the ability to adapt to dynamic changes in traffic flow levels and patterns. Whereas SCATS is more complex to set up SCOOT is not. Both methods however have less robust architecture to failure as they depend on more central machines for analyses and action selection. In these hierarchical and centralized systems the failure of a global or perhaps regional machine would disable a number of intersection controllers. These two systems also lack

the ability to learn. Thus as long term traffic patterns change over time traffic engineers need to be called out to reprogram the systems.

With Girianna and Benekohal's traffic control method (Girianna & Benekohal, 2004) we are introduced to the first method that is able to establish progressive signal systems and has the ability to learn. This approach however is designed to establish progressive signal systems in over saturated networks and the paths of the possible paths of the progressive signal systems throughout the network had to be defined by an expert. Thus the system is relatively complex to set up and it cannot adapt to dynamic changes in traffic flow levels or patterns. Its centralized architecture also reduces its robustness to failure. We thus do not consider it a suitable solution to SQ1.

The methods developed by Bull et al. (Bull et al., 2004) and Pappis and Mumdani (Pappis & Mumdani, 1977) are easy to setup and can both learn over time and adapt to dynamic changes in traffic flow levels and patterns. Neither method however enables the creation of progressive signal systems. Both of these approaches are somewhat similar to the actuated method, only that these methods can learn over time.

The Organic Computing method (Tomforde et al., 2008) developed by Tomforde et al. is a very interesting case with regards to RQ1. It has the ability to create progressive signal systems throughout the transport network that can adapt to changing traffic flow levels and patterns. It also has the ability to learn over time and has a robust distributed architecture. The paths of the progressive signal systems do not have to be set up by an expert but chan change dynamically. The issue with this method however is that in order for it to work it needs to have a microscopic simulator set up per intersection that has an accurate model of the transport network being controlled. This not only makes for a very complex set up process but also insinuates quite high memory and processing requirements for the devices on which the traffic controllers are running. This unfortunatly is somewhat too steep a requirement for our needs and is thus not considered a suitable solution to RQ1.

Srinivasan et al. present a ANN learning based method (Srinivasan et al., 2006) of traffic control that is able to create progressive signal systems throughout the transport network through coordination between the intersection agents. This system can also adapt to changes in traffic flow levels and patterns. This method however has a hierarchical architecture which reduces its robustness to failure. This architecture also makes it more complex to set up as the hierarchy of agents needs to be specified and the network needs to thus be manually devided into regions in which local intersection agents report to agents that are more senior in the hierarchy. This type of system also has more complexity in coding as the supervisor agents and the local intersection agents need to be coded separately. Thus this method is not suitable as a solution to RQ1.

Kosonen presents a traffic control method (Kosonen, 2003) that learns using fuzzy logic and that is able to adapt to dynamically changing traffic patterns. It is relatively easy to set up and has a robust

distributed architecture. It enables coordination between intersections yet it does not enable the explicit establishment of progressive signal systems. Without this ability it is not a suitable solution to RQ1.

The method developed by Bazzan (Bazzan, 2005) is an interesting example as it enables the establishment of progressive signal systems yet it does not require communication or coordination between the intersection agents. It is able to achieve this by using the unrealistic assumption that each intersection agent receives a global reward as opposed to a local reward. Thus the agents have the ability to know how the affects that their actions have on the transport network as a whole. This implicitly leads to progressive signal systems that can adapt to dynamic changes in traffic flow patterns. The assumption that all agents have access to a global reward however is unrealistic in real life, meaning that this method too is unsuitable as a solution to RQ1.

The next seven traffic control methods presented in Table 3 (Prashanth & Bhatnagar, 2011) (Porche & Lafortune, 1997) (Thorpe, 1997) (Camponogara & Kraus, 2003) (Abdulhai et al., 2003) (Wiering, 2000) (Y. Zhang, Xie, & Ye, 2007) rate very similarly with regards to our requirements but have been included in this thesis to show the variety of RL methods that have been applied to traffic control. Each of these methods learn over time using RL algorithms, can adapt to changes in traffic flow, are relatively easy to set up, and have distributed architectures (with the exception of Prashanth and Bhatnagar which has a centralized architecture). None of these methods however take coordination into account and none of them can establish progressive signal systems. Thus none of these methods are suitable as solutions to RQ1.

The remaining five methods presented in Table 3 (Salkham & Cahill, 2010) (Dusparic, 2010) (Bazzan, de Oliveira, & da Silva, 2010) (Richter, Aberdeen, & Yu, 2007) (Kuyer, Whiteson, Bakker, & Vlassis, 2008) again rate very similarly with regards to our requirements. They learn using RL based algorithms and can all adapt to changing traffic conditions. All have distributed architectures and are relatively easy to set up (with the exception of Bazzan et al. which has a hierarchical architecture and is more complex to set up). None of these methods explicitly take into account the establishment of progressive signal systems. Many of the traffic control methods that we have presented, such as these 5, are not able to establish progressive signal systems despite the fact that they allow intersection agents to coordinate their actions with each other. One major telltale sign that these works that do not explicitly address progressive signal systems cannot establish such systems implicitly is when cycle lengths vary from one intersection to another. Thus if the agent is able to change its phase lengths independently of its neighbors, resulting in varied cycle lengths throughout the network, then a progressive signal system cannot be established except by chance. Phase skipping is also a telltale sign of lack of the ability to create progressive signal systems throughout the network, even if the agents coordinate their actions with each other.

In conclusion we have found that although we have investigated a wide variety of traffic control methods, and although some of them can establish progressive signal systems and can learn over time,

none are suitable solutions to RQ1. The method that we have investigated and discussed that perhaps comes closest to fulfilling our requirements is perhaps the Organic Computing method developed by Tomforde et al. (Tomforde et al., 2008). Even this method however has limitations that render it unsuitable as a solution to RQ1. It thus becomes our task to develop a method, namely Qoordination, that enables the creation of progressive signal systems within a transport network through coordination. This method must be able to adapt to dynamically changing traffic flow levels and patterns. It must also have a robust distributed architecture and be relatively easy to set up. Thus Qoordination bridges a significant gap in the current literature. In the next chapter we describe in detail the design of such a system.

2.4 Summary

In this chapter we have introduced the reader to the background information and research relevant to put the work of this thesis into context. We have introduced the concept of intelligent rational agents and have described the elements of agent based learning that are most relevant to this thesis. We then presented relevant traffic engineering concepts and terminology. After having described a number of classical approaches to traffic control we presented a selection of AI based traffic control methods. Particular emphasis was put on methods of traffic control that are based upon the learning based methods presented earlier in this chapter. We have also discussed the positioning of our research that is presented in this thesis within the context of the background information and other relevant research.

Chapter 3

Qoordination

In this chapter we present the Qoordination approach to traffic controller coordination. Qoordination models intersections within a transport network as individual Q-Learning agents that can each optimize their own local traffic flow using any one of a variety of different traffic control methods. Such methods include Round Robin, SAT, and other learning based optimization methods. Qoordination establishes and maintains dynamic progressive signal systems along the main traffic corridors that run through the transport network.

This chapter is organized as follows. We begin by describing a set of requirements to be satisfied by an efficient learning based approach to intersection agent coordination. We then present the motivations behind the Qoordination design decisions. We then give a detailed description of Qoordination and of its unique Multi-Layer Hashing (MLH) utility function.

3.1 Requirements

Through analysis of the information presented in the previous chapter a number of requirements have been identified that act as guidelines in the design of our learning based approach to intersection agent coordination. These identified design requirements are as follows:

- **Requirement 1 (Req1): Coordinated** – this approach must be able to establish coordination among intersections in such a way as to establish progressive signal systems along the main traffic corridors that run through the transport network.

- **Requirement 2 (Req2): Adaptive** – this approach must be able to maintain these progressive signal systems despite changes in traffic flow levels and dynamic changes in the direction and course of the main flows of traffic. An expert traffic engineer should not need to be called out to recalibrate the system whenever traffic patterns change.
- **Requirement 3 (Req3): Rapid Learning** – this approach must be able to learn rapidly in order to keep up with such a dynamic environment. A function approximation technique is required that can perform generalization as well as abstraction so as to allow for rapid learning.
- **Requirement 4 (Req4): Autonomous** – this approach must enable the autonomous emergence of such progressive signal systems based upon the actions taken by autonomous intersection agents in response to current flows of traffic. The progressive signal systems should not need to be manually specified by an expert traffic engineer.
- **Requirement 5 (Req5): Flexible** - this approach must allow for ease of expansion and modification of the transport network topology.
- **Requirement 6 (Req6): Robust** - this approach must be robust to noisy sensor readings such as are unavoidable with traffic sensors like induction loops. It must also be robust to communication failures and avoid single points of failure and data bottlenecks.
- **Requirement 7 (Req7): Scalable** - this approach must be able to be scaled up from use in a simple two intersection arterial up to use within very large transport networks.

In the remainder of this chapter we present the design of the Qoordination approach to intersection agent coordination. We then analyze how this design fulfills the requirements specified above.

3.2 Motivations

In this section we explain the motivations behind the choices made when faced with decisions throughout the design process. Many of these design decisions have an obvious choice due to observations from the research that has been reviewed in the previous chapter. Other decisions do not have an obvious choice as previous research has not provided a suitable option. In these situations new and innovative solutions are required.

3.2.1 Architecture

The architecture of a traffic intersection coordination system can be centralized, hierarchical, or fully distributed. A centralized approach has the benefits of ease of management and a global view of the entire system from a central controller. Although this approach has high computation and memory requirements on some central machine this can be overcome with current advances in hardware and cloud computing.

This approach however does have the drawback of over dependence on communication with a single point of failure. This approach is thus not robust to communication failures or even to communication latency. A hierarchical approach significantly reduces this issue by grouping the intersection controllers into areas, each area having a single processing unit that supervises the area's intersection controllers. This hierarchical approach removes dependence on a single point of failure, which sidesteps the network wide failure that can occur when the central processing unit goes down. It does however still present a point of failure within each area. Another issue that we remarked with this approach was that in all likelihood each area needs to be manually defined by an expert. Inter-area coordination is also an additional challenge. It is likely that a separate set of software algorithms needs to be designed for the supervisor functionality to those designed for the controller functionality. A fully distributed architecture on the other hand has the potential for the autonomy required to fulfill our project requirements thus far. Each intersection within the transportation network can be modeled as an individual autonomous agent. Autonomous agents need not be manually grouped by their area but have flexibility in deciding which other agents within the network to rely on. Each agent can also be run on the same set of software algorithms. An autonomous intersection architecture is thus robust, and flexible.

3.2.2 Technology

Having chosen a fully distributed, agent architecture our next challenge was to investigate the different technologies that could be used to coordinate the agents. As has been shown in section 2.3.3 many different technologies have been applied to traffic control in efforts to enable adaptability to changes in traffic flow. Some of these approaches include fuzzy logic (L. Zhang et al., 2012), neural networks (Srinivasan et al., 2006), ant colony optimization (Putha & Quadrifoglio, 2010), swarm intelligence (de Oliveira et al., 2004), constraint optimization (de Oliveira et al., 2005), genetic algorithms (Girianna & Benekohal, 2004), organic computing (Prothmann et al., 2008), and RL (Camponogara & Kraus, 2003) (Salkham et al., 2008). Wheras these approaches focus on the optimization of traffic flow the work of this thesis focuses on the coordination of agent actions so as to form progressive signal systems across arterials of intersections, which in turn has a positive effect on traffic flow opitimization. The intersections within Qoordinated arterials may be implementing a variety of different traffic flow optimization methods, potentially including some of the ones just mentioned. Of the technologies that we researched we found that RL is the most suited to the purposes of Qoordination agents. An RL agent has the ability to be initialized without any knowledge or model of its environment and then learn through interacting with that environment how to maximize some long term reward. An RL agent has the ability to adapt from an initially blank state to one in which it understands how the environment works and how its own actions affect the environment. This was indicative that RL had the

potential to satisfy our derived requirements of robustness to failure and latency, flexibility for expansion and modification, and adaptability to changing traffic patterns over time.

3.2.3 Environment

The next challenge is to model the agent's environment. The different variables to take into consideration when modeling the agent's environment are given in section 2.1. The approach that we took was to model the agent's environment as a multi-agent, fully observable, stochastic, sequential, discrete, dynamic, unknown environment.

Transport networks are **multi-agent** environments due to the fact that adjacent intersections can be highly coupled. Actions taken by one intersection agent can have a major influence on the rewards received by downstream intersections in light to medium traffic flow levels, or can even have a major influence on the rewards received by upstream intersections in heavy and saturated traffic flow levels.

We assume that the agent's environment is **fully observable**. An environment is fully observable if the agent's sensors can detect all aspects of the environment that are relevant for the agent to choose the appropriate action to take. In the case of our intersection agents this means that the agent knows when a vehicle has arrived on an approach to the intersection and also that the agent knows which neighboring agents are adjacent to it. Although we do assume that the agent knows when a vehicle has arrived on an approach to the intersection this is not fully possible in reality. In practice a sensor is embedded a few meters (e.g. about five meters) before the traffic light with potentially an additional sensor embedded about 30 meters upstream. This means that the agent can detect approximately five average length vehicles at a standstill on any one of the intersection's approaches. Any more vehicles than five that arrive on that approach will go undetected. For an intersection agent to be partially observable it could have an approach that does not have a functioning traffic sensor or it could have an uncontrolled intersection as an adjacent neighbor. This leads to traffic patterns that can be perceived as random, as dynamic, or as errors, but that would be correctly interpreted if the agent could observe their cause. Although partial observability within MDP based systems is a challenge that is receiving significant attention within scientific literature we omitted it from our approach because it is not within the scope of this research. In order to make our assumption of full observability we simply insure that all intersection approaches have functioning sensors and that the evaluation is done within a grid network consisting uniquely of controlled intersections as is common among simulation based traffic control system evaluations.

Traffic by nature is **stochastic**. Just because we reduce the length of a phase and observe a specific reduction in queue lengths as a result does not necessarily mean that we will observe the exact same reduction in queue lengths the next time the action is performed from the same state. Random variances are thus expected within the environment and the coordination approach should be designed so as to

handle this. RL handles stochasticity quite well by implementing a learning rate variable (see equation 12). This ensures that utility values do not change too quickly with stochastic readings but that they settle on more stable average values.

The transport network environment is by its nature a **sequential** process as the transition from one state to another is dependent on the action taken by the agent. Thus an agent can observe temporally discounted rewards over time which is not possible in an episodic environment.

With regards to the design decision of whether to model the state and action spaces as **discrete** or as continuous variables the answer was not immediately apparent, as it might be within a robotics system for example. Traditional RL state and action spaces are modeled using a lookup table, which has a discrete set of possible states and actions. We had initially hoped however to model the state space using continuous variables. This would have required a function approximation mechanism that enabled such functionality. The state variables that are commonly used within coordinated traffic control systems include average waiting times, vehicle throughput, and queue lengths. Each of these are easily bounded between maximum and minimum values. As it turns out, the difference between an average waiting time of 6 seconds and an average waiting time of 6.4 seconds is not that significant within a stochastic traffic networks. The state variables being used thus do not require the level of detail afforded them by continuous representation. We had also initially wanted to represent the action space using continuous values. As an example of this an agent's actions could have included extending a particular offset by 2.5, 3, 3.5, 4, 4.5, or 5 seconds. After much experimentation we discovered that the level of detail given by continuous representation of the actions did not make a significant difference when compared to discrete representation. In fact even changes of one or two seconds do not make too much of a difference within such a stochastic environment. Thus we found that our initial assumption that continuous representation of state and action space was in fact unnecessarily complicating an approach that could work even more effectively using discrete representation. We go into further detail on this particular design decision in subsection 3.2.6.

The agent's environment is modeled as a **dynamic** environment because not only do traffic conditions change regularly even without an agent's interaction e.g. morning and evening peak time traffic, but also longer term traffic patterns change over time. Such changes include alterations in a main traffic corridor's course and direction. Adaptation to these types of changes fulfill the initial requirements stated in section 3.1.

We assume that the agent's environment is initially **unknown** i.e. that the agent does not have a pre-defined model of how the environment works and how the agent's actions affect the environment. This assumption is made because otherwise a pre-defined model would have to be created for each of the individual agent intersections. This would make scaling to a large transport network quite a tedious and error prone process.

3.2.4 Learning Algorithm

The environemnt in which the Qoordinated intersection agent will reside has now been defined, the next step is to decide which algorithm to use for learning. The 'unknown' characteristic of the environment definition in the previous sub-section rules out Dynamic Programming (DP) approaches as they require accurate predefined models of the environment (see section 2.2.2). We are thus essentially left with a choice between Adaptive Dynamic Programming (ADP) or Temporal Difference (TD) algorithms (see section 2.2.2). Although the more popular approach among the two is currently TD algorithms the decision is not an easy one to make. Both approaches have both advantages and disadvantages with regards to the environment that we have now defined. TD approaches tend to be more popular because of their lack of dependancy on a model of the system. This reduces complexity as well as memory requirements because a transition model of the environmnet need not be maintained at all. This also reduces computational requirements during the action selection and learning processes as explained in section 2.2.2.3. One often overlooked aspect of TD however is that although it is less complex, more easily maintainable, and more concise, it is slower to learn than ADP. In dynamic environments, including environments that appear to be dynamic due to their being populated by multiple learning agents, rapid adaptation to change can be a critical factor. The question to be asked is thus whether or not this fact is relevant to our Qoordination approach. Results obtained from our initial experiments (Fagan & Meier, 2014) would suggest that it is. In these initial experiments we applied both ADP and TD learning approaches to an ACRL traffic flow optimization agent within a basic traffic ballancing simulator. When executed within a single agent environment the difference between their performances was negligable, with ADP performing just over 1% better than the TD approach, as can be seen in Figure 29.

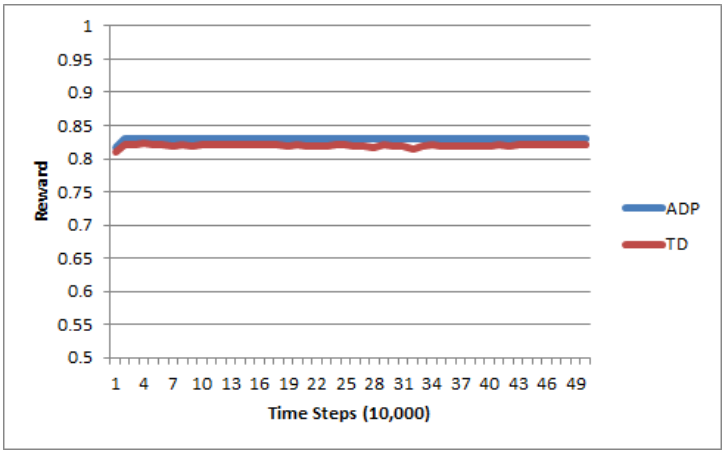

Figure 29 Initial evaluative comparison between ADP and TD approaches in single-agent environment

When executed within a multi-agent environment the difference between their performances was quite substantial, with ADP performing almost 20% better than the TD approach, as shown below.

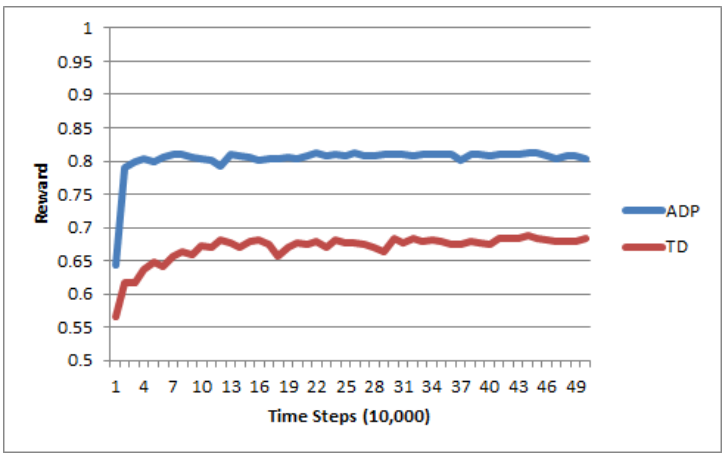

Figure 30 Initial evaluative comparison between ADP and TD approaches in multi-agent environment

These experiments however were conducted within the author's own basic traffic ballancing simulator and it is not immediately clear as to whether a similar effect would take effect within a realistic evaluation. Such findings would be quite significant however an implementation and comparison of both approaches is beyond the scope of this thesis. Although research into ADP based traffic controller coordination warrants research we have decided to incorporate TD learning into our design because of its simplicity, lack of dependance on a model, and lower processing requirements. An implementation of ADP based coordination as well as its comparison to TD based coordination is thus left for future research.

3.2.5 Policy Definition

The next design decisions that we were faced with was whether the learning algorithm should be on-policy or off-policy (see section 2.2.2.3.1) and whether the policy should be implicit or made explicit (see section 2.2.3.1). On-policy approaches learn relative to the current policy and have the benefit of fast and accurate learning. Off-policy approaches learn relative to a greedy policy, even if the current policy is not greedy, thus having the flexibility of being able to learn which actions yield the highest long term rewards even when following explorative actions. Explicit policies have the benefit of being able to handle very large and even continuous action spaces by essentially tranforming the agent into a reflex agent (see section 2.1) that can perform rapid action selection. Learning to optimize an explicit policy however is a longer process than learning to optimize an implicit one (Fagan & Meier, 2014). SARSA (see section 2.2.2.3.2) and ACRL (see section 2.2.2.3.3) are both on-policy approaches. While SARSA has an implicit policy ACRL makes its policy explicit. Q-Learning (see section 2.2.2.3.1) is an off-policy TD algorithm with an implicit policy.

As our Qoordination approach needs to be sufficiently flexible to learn how to minimize vehicle queues even during exploration and when being led by non-optimal policies we have chosen to pursue an off-policy approach.

The decision as to whether or not to make the policy explicit was not a trivial one to make. As we mentioned in section 3.2.3 we had initially intended the action space to have a continuous form of representation. We performed experiments within a basic traffic balancing simulator (Fagan & Meier, 2014) and found that having an explicit policy could lead to higher rewards. This is particularly true because of the high probability of introduced inacuracies in setting discrete action values in the implicit representation e.g. having a granularity of three seconds for action values when the true optimal action value is not a multiple of three. Learning to achieve these higher rewards however has a temporal cost, as can be seen in Figure 31 where it takes approximately 20,000 time steps before ACRL begins to achieve high rewards.

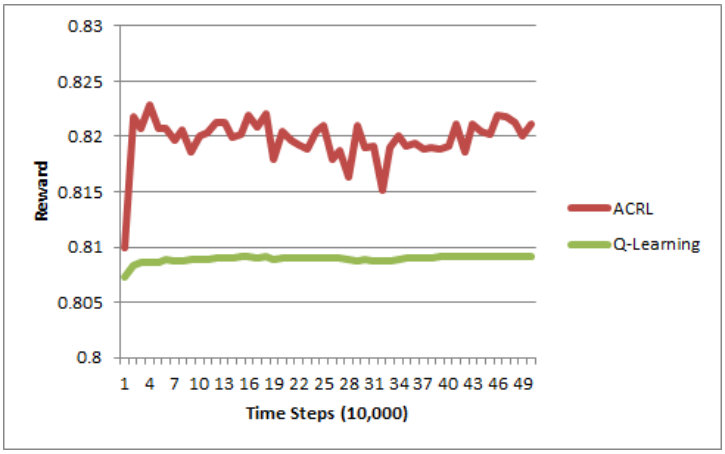

Figure 31 Initial evaluative comparison between TD learning approaches in single-agent environment

We found through our initial experimentation that when executed within a multi-agent environment this increase in learning time actually led to the explicit policy approach achieving lower rewards than the implicit policy approaches, as can be seen in Figure 32.

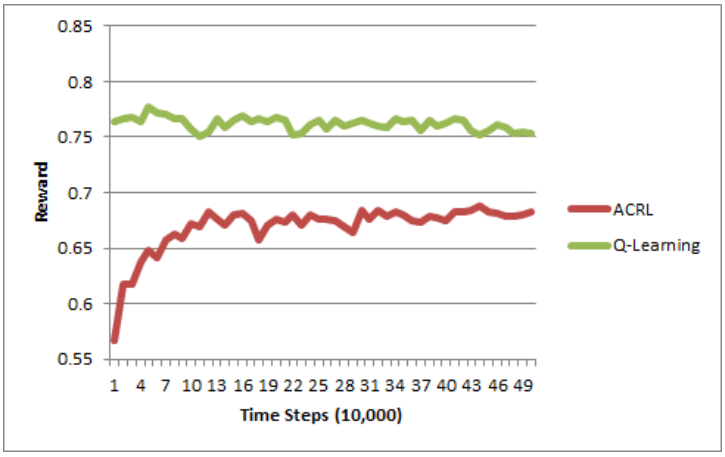

Figure 32 Initial evaluative comparison between TD learning approaches in multi-agent environment

These initial results lead us to speculate that the extra learning time required by explicit policy representation might not be as suited for a multi-agent environment as the implicit policy approach. As mentioned in section 3.2.3 we eventually decided to use a discrete form of representation, which further reduced the need to make the policy explicit. Even using a discrete representation the state space can get quite large in a multi-agent system. As we will cover in sub-section 3.2.8 our approach uses an abstraction method that significantly decreases the state space size. Thus we decided to benefit from the more rapid leaning abilities of implicit policy representation as our action space size is concise enough to benefit very little from the reflex like action selection speed of explicit policy representation.

As the learning algorithm that is to be used by our coordinated agents is to be an off-policy algorithm with implicit policies we can see that the algorithm that will be used will be Q-Learning.

3.2.6 Representation

One of the first major challenges that we identified while reviewing state of the art literature is the fact that RL agent state spaces grow exponentially as the number of agents with whom they coordinate increases.

In order to continue representing utility functions with lookup tables within a multi-agent environment some major abstraction have to take place (Ponsen et al., 2010), eliminating all state variables that do not have a significant impact on the agent's ability to make the right decisions. This approach has the benefit of requiring very few alterations in the single agent learning algorithms presented in sections 2.2.2.

3.2.6.1 Particle Filtering

Our first attempt at discovering a suitable representation for utility function involved using particle filtering to discover the relative importance of each state variable for an agent to select an appropriate

action. Another of our objectives with this method was to investigate the integration of uncontrolled intersections into the system, modeling them as hidden nodes within a hidden Markov Model. This would have modeled the environment as a partially observable environment. We soon found however that this approach was unsuitable in our situation due to the extreme rate of state space expansion with increased network sizes. It was at this point that we decided to consider only controlled intersections and to model the environment as being fully observable.

3.2.6.2 Neuro-Fuzzy Network

From lessons learned in our first attempt we decided that the representation of the utility function should be concise and be able to generalize well. Parametric function approximators (see section 2.2.4.2.1) are very concise and are well able to generalize to situations that they have not yet encountered. Our second attempt at discovering a suitable utility function representation thus utilized a Neuro-Fuzzy Network (Xie, 2007). This approach to function approximation combines fuzzy logic with Artificial Neural Networks (ANN) in an attempt to concisely represent a utility function in a scalable fashion. We found during our implementation of this approach that it was quite complex and required many parameters to be set manually. For this approach the ANN topology needs to be tailored for each intersection agent. As discussed in section 2.2.4.2.1.2 this can be potentially accomplished by training the agents at each intersection a number of times and then selecting the best outputted network. This however does not scale well as it requires an expert to oversee the initialization of the function approximator for each of the individual intersection agents. This can be an error prone and costly process. The fuzzy parameters also need to be set either manually or by some additional training process. This approach also requires major modifications to the single agent learning algorithms presented in sections 2.2.2. We also found that the addition of fuzzy logic to the ANN did not increase the systems scalability as much as we would have hoped for. Our overall outtake to this approach was that it was overly complex and required too much manual intervention. We also found that it was not scalable enough to be considered a suitable approach for our needs.

3.2.6.3 Artificial Neural Network

After seemingly fruitless experimentation with the previously mentioned Neuro-Fuzzy approach we decided to simplify matters by implementing a more traditional ANN. We based our ANN implementation upon the Encog machine learning framework (http://www.heatonresearch.com/encog accessed on 06/04/2015). This approach took significantly less time to implement than the previous approach. Although this approach did not require the setting of fuzzy parameters it still did require calibration of the network topology, which would need to be done individually for each intersection agent. It also still required significant modifications to the learning algorithms presented in sections 2.2.2.

An optimal approach for our purposes should require little or no manual parameter setting, calibration, or intervention in training.

3.2.6.4 High Dimensional Clustering

Our next attempt at representing the utility function took a non-parametric approach. Unlike the previous attempts it was hoped that this approach would not require significant modifications to the learning algorithms presented in sections 2.2.2. The specific approach that we took was that of High-Dimensional Clustering (see section 2.2.4.2.2.1). This approach enabled the calculation of the relative importance of each state variable in an agent's ability to choose an appropriate action. The state variables were then weighted appropriately and these weights were used in calculating the expected reward for any given action. This approach allowed for a more intuitive extension of the learning algorithms important to our Qoordination approach than the parametric attempts. We found however that this approach was limited in its ability to scale in large state spaces. Each additional state variable included required more calculations to be made for each action selection operation. Also, as a separate record was created and stored at each time step this would eventually lead to significantly large storage and processing requirements. Thus although this approach was more intuitive to implement as a function approximator it did not scale well enough for us to consider it as suitable for our needs.

3.2.6.5 Multi-Layer Hashing

In Russell and Norvig's book "Artificial Intelligence, a modern approach" (Russell & Norvig, 2010), a description of the Locality-Sensitive Hashing technique that is described in section 2.2.4.2.2.2. With this technique as an inspiration and basis we developed a hash based function approximator that fulfills all of our requirements. It is a non-parametric approach which allows for the intuitive extension of the learning algorithms important to our Qoordination approach. It enables generalization and very few calculations for its roles in learning and action selection. It also enables automatic weighting of state variables in accordance to their importance to agent's ability to choose an appropriate action. This allows for abstraction. The details of this hash based method of function approximation, which we refer to as Multi-Layer Hashing (MLH), will be given in section 3.3.

3.2.6.6 Summary

Having experimented with a number of methods of function approximation we found that a method of representing the utility function that was suitable to our needs was not available. We thus developed the hash-based MLH method.

3.2.7 Reward Scope

Whereas many approaches reviewed in Chapter 2 illustrate agents optimizing local rewards i.e. selfish behavior (Camponogara & Kraus, 2003) (Abdulhai et al., 2003), others optimize global rewards i.e. selfless behavior (Bazzan, 2005). Our Qoordination agents calculate local rewards based upon the vehicle queue length information that is available to them. One particular question that arose during Qoordination design was whether it would be beneficial to combine adjacent neighbor rewards with the agents' local reward. In order to answer this question we implemented Qoordination agents that maintained a separate utility function for each of their adjacent neighbors as well as for themselves. Thus an agent was aware of how its actions affect itself and each of its neighbors. It was not however aware of any of the effects of any of its neighbors' actions. Each utility function was then used for a separate implicit policy. Having multiple policies the agents then used an arbitration based multi-policy approach to choose the actions that would lead to the highest negative effect if they were not chosen. This arbitration approach is similar to W-Learning (see section 2.2.3.2.2). We found that each agent could accurately distinguish how to act so as to optimize either their own rewards or those of any of their neighbors. We also found that they could choose whether it was more important to optimize their own reward or to optimize the reward of one of their neighbors from any given state. We found however that each agent's constant change from selfish to selfless behavior lead to confusion in the interdependent learning processes. Learning times were thus increased dramatically. We found that consistently selecting selfish actions i.e. ones that optimized an agent's own local reward lead to much more reliable results and much faster learning times.

It is important to note that our Qoordination approach consists of agents that are not only non-competitive i.e. they are not designed to benefit from the loss of any other agent, but that are cooperative. Thus they are not deceptive in the information that they share with neighbor agents or with the actions that they perform.

3.2.8 Irrelevant Agent Abstraction

High coupling of intersection agents can result in exponential increase in state space sizes with every additional adjacent agent. This does not scale well. One solution to this is to use a suitable function approximator to handle the increasing dimensionality (Prashanth & Bhatnagar, 2011). This is scalable to a certain extent but does have limits. Another approach is to use abstraction techniques (Ponsen et al., 2010) to reduce the number of other agents that an agent is aware of in its environment and hence reduce the state variables being included. We decided to abstract away other agents within the network in a similar fashion as is done with coordination graphs (see section 2.2.6). This is done by having agents coordinate exclusively with other agents that are directly influenced by their actions or whose actions directly influence them. We have decided that intersection agents will only communicate with their immediately adjacent neighbor intersection agents. This implies that an intersection is directly influenced

by and directly influences adjacent intersections. This drastically reduces the size of an agent's state space. A Qoordination agent's state space will thus consist of state variables that require only local data and data from the intersection's immediate neighbor intersections. This also means that agents do not need to be divided into regions or groups but that they each belong to a set of overlapping one-hop groups, as is illustrated in Figure 33, with controller agents at intersections A and B participating in a total of five separate groups each.

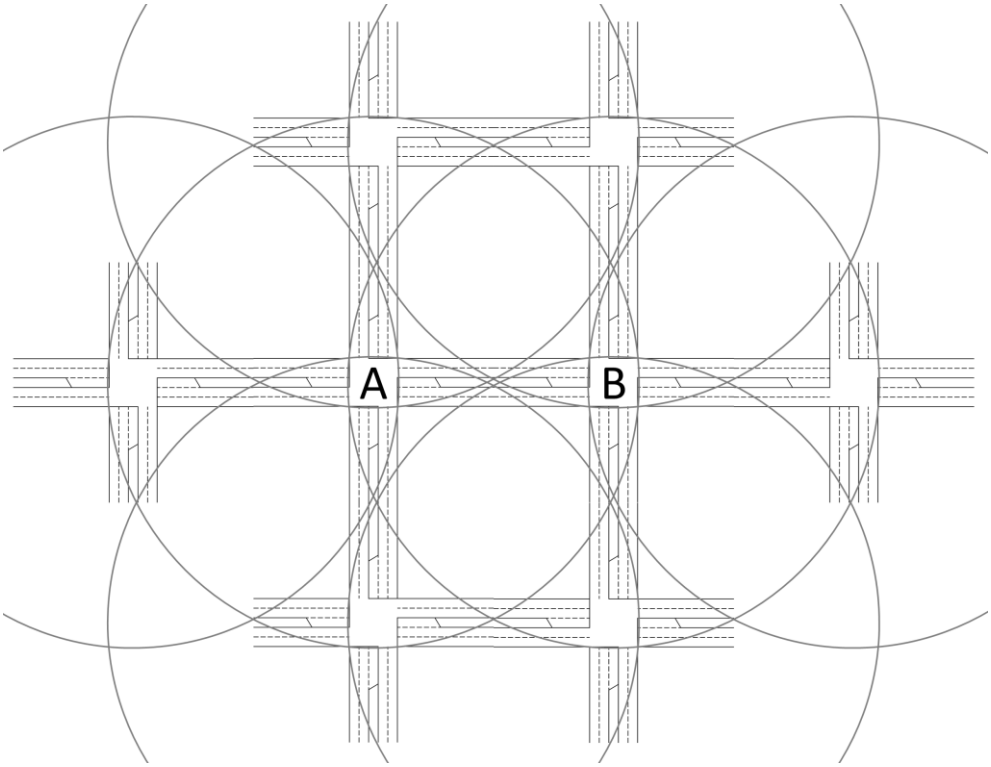

Figure 33 Example of overlapping one-hop neighbor groups

Not every intersection is dependent on all of its adjacent neighbor intersections. For example, in low to medium traffic flow levels an intersection will not be effected by the actions of downstream neighbor intersections. Learning times can be reduced if non-essential adjacent intersection agent data is removed from consideration. Irrelevant state variables can be abstracted from the agent's state space using the MLH function approximator that we describe in section 3.3.

3.2.9 Exploration Scheme

An appropriate exploration scheme is essential for a learning agent's long term benefit (see section 2.2.5 for exploration scheme examples). Exploration enhances an agent's performance by providing it with a more complete and more updated understanding of its environment and the effects of its actions within this environment. Throughout the course of our design and development process we experimented

with many forms of exploration schemes including greedy, ϵ-Greedy, Boltzmann, and various combinations of these algorithms. Exploration schemes tend to start with a high likelihood of selecting explorative actions. This likelihood is reduced over time as the utility function's accuracy increases. We have found that in multi-agent environments such as transport networks this approach to exploration is not efficient. This is because interdependent agents learn relative to the actions of the agents on whom they depend. If all agents begin by selecting relatively high numbers of random actions then each one only learns the utility of performing actions relative to neighbor actions that do not actually optimize the neighbors' rewards. This tends to lead to confusion. We have thus designed a novel approach to exploration within multi-agent environments that takes this relative learning into account. This novel exploration scheme is initialized with a very low ϵ value. This value is slowly increased until it reaches a maximum threshold level or until a random action has been selected for execution. Once a random action has been selected the ϵ value is again reduced to its low initial state. This process of slowly increasing the ϵ and then dramatically reducing it to its low initial state continues throughout the learning process.

3.2.10 State

Qoordination aims to find the offsets between adjacent intersections that enable the establishment of progressive signal systems along the main traffic corridors that run through the transport network. It stands to reason that the offset values between the intersection and its adjacent neighbor intersections are included as variables in the state space. These offset values are calculated by the agent requesting from its adjacent neighbors to know how many seconds remain until the start of their next cycle. The time remaining until the start of the intersection's own cycle is subtracted from this and if the result is a negative number then the neighbor intersection's cycle length is added to it.

A second variable that is to be taken into consideration for a Qoordination agent's state is that of the main direction of traffic flow. This can be easily calculated by comparing the vehicular throughputs of each of the intersections approaches. Possible directions are North, North West, West, South West, South, South East, East, and North East. Thus optimal offsets can be set for each possible traffic flow direction.

Another variable that one would consider would be that of vehicular throughput or some other variable that measures traffic congestion at an intersection. Such a variable would be important to distinguish between normal traffic flow and congested / over saturated traffic flow. We have found that optimal offset values along a traffic corridor remain the same through light, medium, and heavy traffic flows. Only in the case of fully congested conditions would we need to switch from forward progression to reverse progression. Traffic congestion stand stills tend to occur when the traffic flowing into an approach is more than what can flow out. This could happen if an upstream intersection has a higher phase length than the phase length that lets the released traffic through the intersection. It could also happen if multiple upstream intersections feed into a single approach via an un-signalized intersection.

These situations are not taken into account in our evaluations and thus a measure of the traffic flow levels is not included as a Qoordination state space variable.

3.2.11 Action

Qoordination is a method of coordinating the actions of intersection agents along dynamic traffic corridors such that they form and maintain a progressive signal system. This is done by adjusting the agents' offset values. Adaptation of other parameters such as phase lengths and cycle length are dealt with by the different traffic control methods being implemented by the intersection agents. Qoordination deals expressly with coordination and thus only modifies the offsets between intersections. A Qoordination agent's choice of actions is to either do nothing or to increase or decrease its own offset to its neighbor intersections. It can modify its offset to its neighbor intersections by either increasing or decreasing its cycle length for the duration of one full cycle. This temporary adjustment is spread out evenly among each phase so that they are uniformly increased or decreased in length for that cycle alone. A Qoordination agent can thus only modify its offset to all of its adjacent neighbors at once and cannot modify its offset to any one of them individually. Offset adjustment sizes are a fixed duration that is a multiple of the intersection's number of phases. In our evaluation scenarios each intersection has six phases and the intersection agent can modify its offset values by 12 seconds at a time. We found that modifications of 6 seconds were often too small to register a significant change in state once the offset modification actions had been performed.

3.2.12 Reward

Possible options for Qoordination rewards include: increased throughput, increased average vehicle speeds, decreased vehicle waiting times, decreased queue lengths, or some combination of these variables. All rewards considered are averaged over the course of the cycle or specified number of cycles between updates (see section 4.2.2.2). These immediate rewards are then applied to the update algorithm (see section 3.3.7) at the end of the time-step in which they are calculated.

3.2.12.1 Throughput

Throughput optimization attempts to get as many vehicles through the intersection as possible. This however can come at the cost of long queue lengths. An agent using this form of reward does not appreciate having an intersection void of traffic, but seems to rather have traffic queues present to increase throughput rate. The goal of Qoordination is to establish and maintain dynamic progressive signal systems along the main traffic corridors that run through a transport network. It is thus not explicitly of concern to Qoordination how many vehicles pass through the network but just that the vehicles stop as little as possible from the time they enter the network to the time they leave it.

3.2.12.2 Vehicle Speed

Detecting vehicle speeds is one of the more inaccurate and difficult options to achieve. This is simply due to the nature of the induction loop sensors. This inaccuracy can be reflected in rewards based upon such a variable.

3.2.12.3 Vehicle Waiting Time

Vehicle waiting time is the sum amount of time that vehicles that are waiting at an intersection have been waiting there for. Caution should be taken when calculating the waiting time as otherwise some unexpected results may arise. For example, an approach that has two vehicles in the queue which have been waiting there for 4 and 6 seconds has an average waiting time of 5 seconds. In the next time step another vehicle joins the queue and so the waiting times are 0, 5, and 7. The average waiting time now becomes 4 seconds while if no additional vehicle would have joined the queue then the average waiting time would have been 6. We would have hoped however that the reverse would have happened by having the reward express a worse state with three vehicles in the queue than two. Summing the individual vehicle waiting times per approach does however behave in the way that we would hope. Vehicle waiting time is limited to the number of vehicles that can be detected as sensors can usually only detect the presence of approximately five average length vehicles (see section 3.2.3). Using vehicle waiting times as a reward has an unforeseen drawback. If given the choice between having vehicles arriving too early at an intersection or risking have them arrive too late this variable will select to have them arrive too early. Qoordination agents using this form of reward thus tend to have all vehicles wait for a small amount of time at each intersection so as to not risk any of these vehicles being left behind when the light turns red. This goes against the concept of progressive signal systems.

3.2.12.4 Queue Length

Queue length optimization is also quite limited as a variable to be used as a reward. This is again due to the fact that the maximum length of a queue that can be detected on any one approach is approximately five average length vehicles. Thus one might think that queue length based rewards would work quite well under light traffic flow levels but would lose their effectiveness under medium to heavy traffic flow levels. This limitation however can be overcome by RL's ability to take into account long term rewards. From a state of having full traffic queues actions are given preference that can lead to empty queues, even if that means that another few actions need to be taken before the agent gets to that state. Thus RL's far sightedness overcomes the weaknesses of using queue length as a reward value. Unlike agents that use vehicle waiting time rewards, if given the choice between having vehicles arriving too early at an intersection or risking have them arrive too late intersection agents using queue length rewards tend to

risk having the vehicles arrive too late. A traffic corridor whose intersections minimize queue lengths thus aims to establish progressive signal systems.

3.3 Multi-Layer Hashing

In this section we describe in detail our Multi-Layer Hashing (MLH) approach to function approximation.

3.3.1 Hashing

As stated in section 2.2.4.2.2.2, a hash table is a data structure that maps hash function generated keys to buckets containing values. These structures provide much faster access to stored values than regular lookup tables. In order to generalize data a function approximator needs to be able to find states that are similar to a given one. This requires some form of organized storage whereas hash tables allocate bin space randomly within memory and thus rely on exact key-bin matching. In order for a hash map based function approximator to be sensitive to state similarity its hash function should place utility values of states that are similar to each other into the same bins.

3.3.1.1 State Variable Hashing

Our MLH function approximator calculates the hash value *h* for state variable *i* by integer dividing (\) the state variable value *x* by a specified coarseness value *c* as shown below.

$$h_i = x_i \backslash \mathrm{c}_i \tag{33}$$

The hash value for a state variable with a specified coarseness value of 2 and any state variable value ranging between 4.0 and $5.\dot{9}$ (where $5.\dot{9}$ represents 5.999… with 9 going on through to infinity) is thus 2. A state variable with a coarseness value of 10 and a state variable value between 30 and $39.\dot{9}$ will have a hash value of 3 returned. Following this logic, when the coarseness value is set above the maximum value a state variable can have i.e. that state variable's upper limit, then assuming that the state variable value is always positive, which assumption we make on all state variable values, the hash function will always return a hash value of 0.

3.3.1.2 State Hashing

A state is comprised of a set of state variables, thus the state's hash key *sh* is comprised of a combination of the set of its state variable hash values. The hash function must ensure that this

combination of state variable hash values is non-commutative e.g. $sh(0,3,0) \neq sh(0,0,3)$. It should also ensure that the state variable hash values are unique for the given coarseness values e.g. $sh(0^{c_1^1}, 3^{c_2^1}, 0^{c_3^1}) \neq sh(0^{c_1^2}, 3^{c_2^2}, 0^{c_3^2})$, where $c^1 \neq c^2$. With coarseness values set above the maximum values that state variables can have the hash function will return the same hash key for all states, regardless of their values, thus assigning them all to the same bucket. With coarseness values set very low the hash function will group only very similar states into the same buckets.

3.3.2 Layers of Granularity

The MLH approach gets its name from the fact that it maintains a separate hash table for multiple sets of coarseness values. These coarseness values can be either discrete or continuous values. Minimum and maximum coarseness value are specified, with the minimum being any small positive value, and the maximum being just above the maximum values that the state variables can have i.e. just above the state variable upper limits. The coarseness values for a specified number of layers are then linearly distributed between the maximum and minimum coarseness values.

We will illustrate this with an example. Let us say that we have an intersection with three single lane approaches and a timing plan that consists of two phases. The state space could consist of the queue lengths on each of the three approaches as well as the phase durations of the two phases. The highest queue length that we can measure is that of five vehicles so we can set the maximum and minimum coarseness values for each of the queue length state variables to 5.01 and 0.01 respectively. We can set the maximum coarseness values of the phase length state variables to some high threshold value, such as 100.01 seconds, while the minimum coarseness values for these state variables can be set just above a minimum allowable phase length of 10 seconds i.e. 10.01. This gives us the following set of minimum coarseness values c^1 = [0.01, 0.01, 0.01, 10.01, 10.01] and the following set of maximum coarseness values c^4 = [5.01, 5.01, 5.01, 100.01, 100.01]. If we decide to have a total of four sets of coarseness values then the two additional sets of coarseness values created are thus c^2 = [1.67, 1.67, 1.67, 40.01, 40.01] and c^3 = [3.34, 3.34, 3.34, 70.01, 70.01]. All states that have c^4 applied will thus be grouped into the same bucket, while states that have c^1 applied will be grouped into buckets with other states that are very similar to them.

This multi-layer hashing method is illustrated below. In this illustration the state consists of only two state variables (to allow us to visually represent the hash table as a two dimensional table) and has ten levels of coarseness.

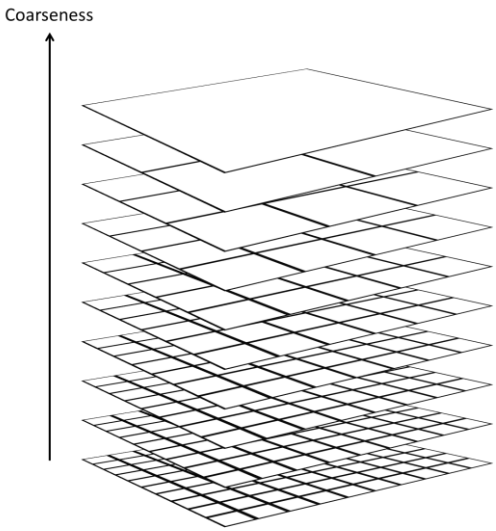

Figure 34 Multiple hash table layers

A state is assigned to exactly one bucket per layer. The reward received for performing an action from a given state updates the values within the state's assigned bucket on each of the MLH layers. This is illustrated in Figure 35.

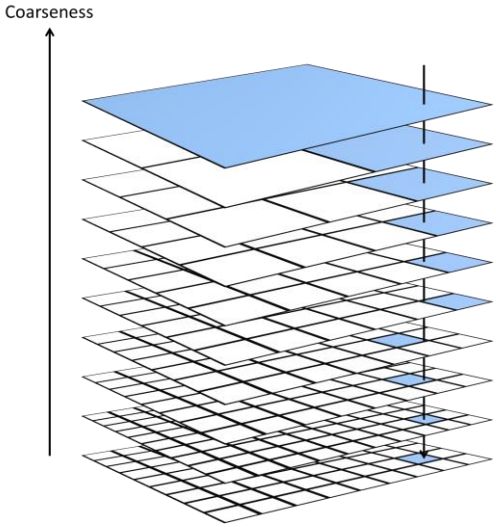

Figure 35 State's single bucket assignment on multiple hash table layers

3.3.3 Inner Action Hash Tables

If each bucket contained a single value then this approach would be suitable for maintaining utility values for only a single action. In order to deal with multiple actions we need to add an additional layer of storage. Within each MLH bucket is thus contained another hash table structure that maps the possible

actions to the stored utility values. The hash keys generated for each action are based upon the action type and the action value, thus modifying a phase length by 5 seconds will generate a different hash key to modifying the same phase length by 10 seconds or modifying a different phase length by 5 seconds. The buckets within this inner action hash table contain not only the action-state utility values but can also contain additional values such as: the expected immediate reward, a counter value to keep track of how many times this particular utility value has been update i.e. an update counter, an error value, or even a transition function for an environment model.

3.3.4 Exploration

The update counter can be quite useful for the purposes of exploration and exploitation balancing. The upper MLH layer update counters increment relatively rapidly and one possibility is to take explorative actions if the update counters are below a threshold level in one of the lower MLH layers. A MLH layer can be considered fully explored when each bucket within each of the layers inner action hash tables has an update counter value above a certain threshold value. This will happen quite rapidly with the upper most MLH layers and will then gradually happen to the increasingly granular MLH layers.

3.3.5 Error

The error value within each inner action hash table bucket represents how accurate the values being stored in that bucket actually are. This error value δ can be calculated using the following equation:

$$\delta = \delta + \mu(\max(r, r_e) - \min(r, r_e) - \delta) \tag{34}$$

where:

- μ is an error learning rate
- r is the reward being used to update the MLH approximator
- r_e is the reward that the MLH approximator would have expected to be returned, usually an average or the most recent of past rewards used to update this bucket's values

This error value will be relatively high for MLH layers that are either too granular or too coarse. In a deterministic environment the lowest MLH layer will have the lowest error rate. In stochastic environments this is not particularly true as stochasticity can make the lowest layers susceptible to outliers. In stochastic environments this error rate thus helps us to identify the granularity level that results in the most accurate values. When the MLH structure is being queried for the utility value of a given state and action the utility for each layer is firstly found. In relatively unexplored MLH approximators a null value will often be returned for a number of the lower layers. The final action-utility

value returned however either can be the most granular non-null value i.e. the value from the lowest layer that does not return a null value, or it could be the value that is associated with the lowest error rate.

3.3.6 Action Selection

The basic MLH action selection algorithm is given below.

Given state s
var finalChosenAction=null
var lowestError=max value
for each layer l
    for each state variable i
        $h_i = x_i \backslash c_i^l$
    $sh = stateHash(h_1^{c_1^l}, \dots, h_i^{c_i^l}, \dots, h_H^{c_H^l})$

    var chosenAction=null
    var highestUtility $= -1$
    for each action a
        ah=actionHash(a)
        if getUpdateCounter(sh,ah)>1
            $Q_l(s,a)$ *=lookupUtility(sh,ah)*
            if $Q_l(s,a)$*>highestUtility*
                highestUtility=$Q_l(s,a)$
                δ_l*=lookupError(sh,ah)*
                chosenAction=a
        else if l>thresholdValue
            return a

    // We can leave out the second half of the following if statement if we want to return the action
    // associated with the most granular layer as opposed to the most accurate layer, assuming that
    // the layers are ordered by descending coarseness values
    if chosenAction!=null && δ_l*< lowestError*
        lowestError=δ_l
        finalChosenAction=chosenAction
return finalChosenAction

Figure 36 MLH action selection algorithm

The basics of this algorithm can be described using the following equation:

$$\pi(s) = \underset{l}{minError} \; \underset{a \in A(s)}{argmax} \, Q_l(s,a) \tag{35}$$

Or if we are to use the utility value given from the most granular layer that does not return a null value we would use this equation:

$$\pi(s) = \underset{l}{mostGranular} \; \underset{a \in A(s)}{argmax} \, Q_l(s,a) \tag{36}$$

This is the equivalent of the following traditional Q-Learning equation:

$$\pi(s) = \underset{a \in A(s)}{argmax} \, Q(s,a) \tag{37}$$

3.3.7 Update Algorithm

The MLH update algorithm is given below.

Given previoiusState s, currentState s', action a, and reward r

for each layer l

for each state variable i of state s'

$h'_i = x_i \backslash c_i^l$

$sh' = stateHash(h'^{c_1^l}_1, \dots, h'^{c_i^l}_i, \dots, h'^{c_H^l}_H)$

var highestNextUtility $= -1$

for each action a'

ah'=actionHash(a')

$Q_l(s', a')$ *=lookupUtility(sh',ah')*

if $Q_l(s', a')$*>highestNextUtility*

highestNextUtility=$Q_l(s', a')$

for each state variable i of state s

$h_i = x_i \backslash c_i^l$

$sh = stateHash(h_1^{c_1^l}, \dots, h_i^{c_i^l}, \dots, h_H^{c_H^l})$

ah=actionHash(a)

updateCounter(sh,ah)++

$Q_l(s, a)$ *=lookupUtility(sh,ah)*

var updatedUtility=$Q_l(s, a) + \alpha(r + \gamma highestNextUtility - Q_l(s, a))$

updateUtility(sh,ah,updatedUtility)

var r_{stored} *=lookupReward(sh,ah)*

updatedReward=r_{stored} *+* α*(r-*r_{stored}*)*

updateReward(sh,ah,updatedReward)

δ_l*=lookupError(sh,ah)*

updatedError=$\delta_l + \mu(\max(r, r_{stored}) - \min(r, r_{stored}) - \delta_l$

updateError(sh,ah,updatedError)

Figure 37 MLH update algorithm

Notice how the action-utility update equation used in this algorithm updates the action-utility using the same Q-Learning equation given as equation 12. This equation is given again below for the reader's convenience:

$$Q_l(s,a) = Q_l(s,a) + \alpha(R(s) + \gamma \max_{a' \in A(s')} Q_l(s',a') - Q_l(s,a)) \quad (38)$$

3.3.8 Layer Abstraction

A fully explored layer is one in which all actions have been performed from within all states in the layer a specified number of times. The least granular layer, which has only one state, becomes fully explored quite rapidly, assuming that all actions are performed at least the specified amount of times. When a layer has been fully explored it becomes possible to remove any fully explored layers above it from the MLH structure. The MLH always returns the utility value of the most granular layer that did not return a null value or of the layer that returns the utility value with the lowest error value. When the second least granular layer has been fully explored as well as the most granular layer, assuming that the second least granular layer utility values all have lower error values than those of the least granular layer then the utility values of the least granular layer will never be returned. The least granular layer can thus be removed. When the third least granular layer is fully explored if the error values are not being used or if all of the error values associated with its utility values are lower than those associated with the utility values of the second least granular layer then the second least granular layer can be removed, and so on. Thus as the environment becomes more fully understood through exploration MLH computation and memory requirements can be reduced. If error values are being used then it is important to consult the error values within each layer during the process of layer removal as the layers that have the lowest error values should not be removed.

3.3.9 Layer Parameter Space Searching

To this point we have presented a MLH function approximator that can handle environments where each state variable is of equal importance. This is not always the case. This can lead to reduced learning speeds and accuracy due to the added complexity of included irrelevant or less-relevant state variables. For example, a non-busy traffic intersection may be heavily dependent on the traffic flow coming from its upstream intersection neighbor, but is perhaps not so dependent on a downstream intersection. In order to accommodate for this the MLH function approximator can identify the relative importance of the different state variables. To this point we have assumed that the initial coarseness values are calculated linearly from given maximum and minimum values for each state variable. The higher the coarseness

values the less relevant are the state variables. Thus with maximum coarseness values set the values of all state variables are irrelevant and are all put into the same bucket. With minimum coarseness values set the values of all state variables become fully relevant. MLH can also however initialize a set of coarseness value sets with random numbers that lie in the range between their maximum and minimum values instead of initializing them linearly. These random coarseness sets are then added into the MLH as individual layers. These randomized layers can then be evaluated by averaging the error values contained within them. Although it is possible to create a layer for each possible combination of coarseness values this would lead to unacceptable processing and memory requirements with increasing numbers of state space variables. A better approach is to randomly initialize a number of MLH layers and then use one of the local search algorithms described in section 2.2.3.1.2.1 to find the layers that have the lowest error rates. With their basic representation i.e. an array of coarseness values, and a suitable evaluation function genetic algorithms could be easily applied to find the most suitable layers to be used. We have found that the simulated annealing algorithm works well in discovering which layers are best discarded and which are best kept. The most accurate layers will assign higher coarseness values to state variables of low importance and lower coarseness values to state variables of higher importance. After a period of training the most accurate layers can be selected for retention and a process of abstraction (see section 3.2.8) can occur in which the state variables that have consistently been assigned coarseness values above a threshold value can be completely removed from consideration.

3.3.10Conclusion

The MLH function approximator that we have presented in this section has the capability of efficient generalization so as to allow for rapid learning. MLH also identifies the importance of each state variable in maintaining accurate data and can automatically abstract away irrelevant state variables so as to increase learning speed and accuracy. Additionally the MLH function approximator can automatically discover the most accurate state variable parameter settings that usually require manual definition. Thus MLH requires much less parameters to be set than other parametric methods of function approximation e.g. manual setting of thresholds to state whether a traffic queue is long, medium, or short, while also requiring no manual topological adjustments that are common in parametric models e.g. how many parameters, nodes, layers, etc. are necessary to represent the data. This approach also allows us to use the RL algorithms such as those described in section 2.2.2 without major modification.

3.4 Process of Qoordination

Qoordination aims to establish and maintain dynamic progressive signal systems along the main traffic corridors that run through a transport network. We will explain such a progressive signal system with an example that is illustrated in the diagram below:

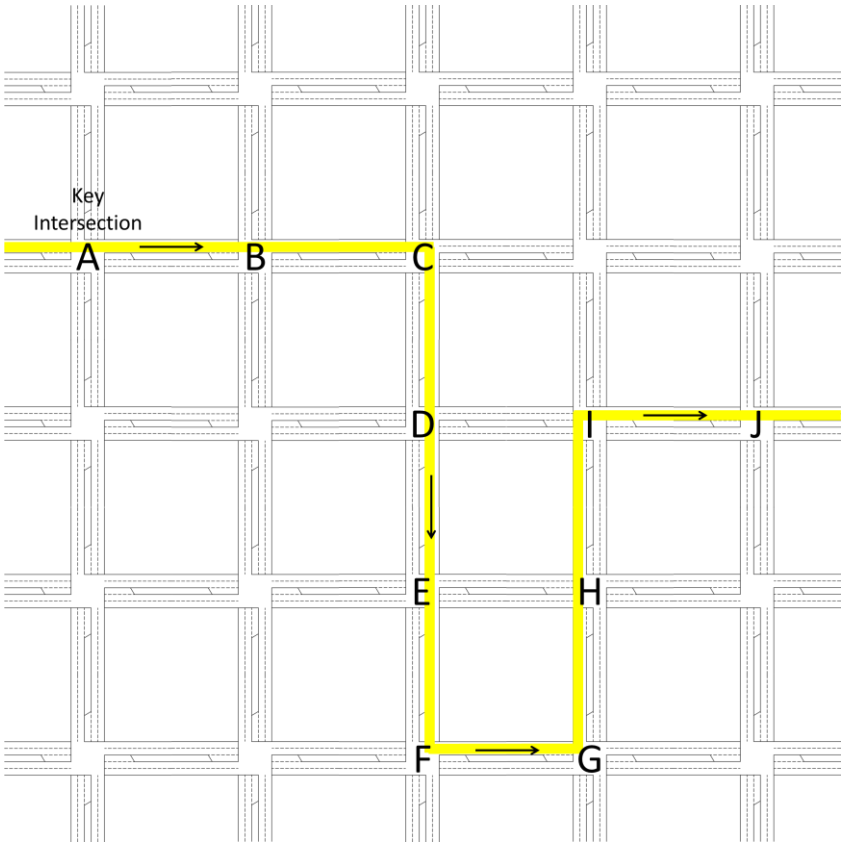

Figure 38 Progressive signal system along a main traffic corridor

In the above diagram the stream of traffic running through this network that has the highest traffic flow levels run along the highlighted path. This path is thus the main traffic corridor that runs through the network. In order to establish a progressive signal system under light to heavy traffic flow levels the traffic lights of the labeled intersections need to turn green in the sequence A, B, C, D, E, F, G, H, I, J. Traffic that is released through intersection A should then be able to progress along the main traffic corridor through to intersection J without having to stop for a red light at any of the intersections along the highlighted path. For the progressive signal system to be dynamic the sequence with which the lights turn green should reverse if the main flow of traffic reverses and begins flowing from intersection J through to intersection A. If the path of the main traffic corridor changes this should also be reflected in the sequence in which the traffic lights turn green.

3.4.1 Detect Key Intersection

The first step in achieving a Qoordinated progressive system is to detect key intersections in the transport network. Key intersections are those through which relatively high traffic flows enter the network. Detecting key intersections within the network would be a trivial process if it could be done in a centralized fashion. In this fashion traffic flows passing through an intersection could be compared to traffic flows passing through any other intersection in the transport network. As Qoordination is a distributed approach to signal coordination the detection of key intersections must be achieved in a distributed fashion. This means that no intersection agent should be required to consider any intersection in the network to which it is not adjacent. As each agent is only aware of itself and its directly adjacent neighbors there will be many agents that do not know about the intersection that is the real key intersection. For example, in Figure 38 intersection agent J does not know about intersection A, which is the real key intersection. A Qoordination agent must thus determine whether it itself is more likely to be controlling the key intersection or whether one of its adjacent intersections is more likely to be controlling the key intersection. In Figure 38 agent J will determine that agent I is more likely to be controlling the key intersection than it is. Agent I will then determine that agent H is more likely to be controlling the key intersection than it is and so on until agent A determines that it is more likely to be controlling the key intersection than any of its immediate neighbors. To achieve this each agent must first create a mapping between its neighbor intersections and its local approaches. The agent thus needs to know where the traffic on each of its approaches is coming from. The agent then determines that its neighbor that is the source of the highest amount of vehicles flowing through it is most likely the key intersection. If the agent's local approach that has the highest throughput (compared to its other local approach throughputs) has no neighbor mapped to it then the intersection itself must be at the edge of the transport network and the agent itself is most likely to be controlling the key intersection. The major challenge with this method of determining the key intersection is mapping the intersection's approaches with the neighbor intersections from which the traffic on these approach flow. This task can be greatly simplified if the intersection is given specific information beforehand. For example, when an agent knows that it is part of a grid network, such as the ones that we use for our evaluations, it is a trivial matter to map each intersections north most approach with the adjacent intersection to the north. In practice however this information is generally not available and even if it were it would not allow for flexibility in change in network topology. There is however an alternative approach to mapping an intersection's approaches with their corresponding neighbor source intersections. For this approach each intersection agent maintains a MLH utility function not only based upon their own rewards but also a separate MLH utility function based upon the rewards of each of their adjacent neighbor intersection agents. These utility functions are formed by requesting adjacent intersection agent rewards during each update. Thus neighboring agents must share reward information among each other. Each intersection agent can now be

aware of the effects of its actions not only on itself but also on each of its adjacent intersection agent neighbors. The agent queries each of the MLH utility functions that it maintains for its adjacent neighbors and discovers for each one the minimum utility value or immediate reward that the neighbors can be expected to receive from any of its actions. These values represent the effect the agent has on its neighbors. The intersection agent then requests from each of its adjacent neighbors to know their effects on it. If the intersection agent has a greater effect on a neighbor than the neighbor has on it then the traffic flow is moving from the intersection to that neighbor. The reverse is also true. This process of calculating the direction of traffic flow between adjacent intersection agents i and j is shown in the equation given below:

$$dir = \min_{a_i \in A_i(s_i)} Q_i^j(s_i, a_i) > \min_{a_j \in A_j(s_j)} Q_j^i(s_j, a_j) \tag{39}$$

where:

- dir is a boolean representing the direction. 1 represents traffic flowing from agent i to agent j while 0 represents the reverse.
- Q_i^j is the utility function that agent i maintains for agent j

If all traffic flows are moving away from the intersection then it is most likely to be controlling the key intersection. Otherwise, the neighbor that has the highest minimum utility for the intersection agent is more likely to be controlling the key intersection. These calculations are performed at every time step so if there is any change in traffic flow direction an agent can change status from being most likely to be controlling the key intersection to having one of its neighbors as being more likely to be controlling the key intersection or vice versa. There are two main differences between an agent that considers itself to be controlling the key intersection and an agent that considers one of its neighbors as being more likely to be controlling the key intersection i.e. a non-key agent. An agent that considers itself to be most likely to be controlling the key intersection will not adjust its offset. It will also be the agent that defines the cycle length that will be used along the traffic corridor.

As transport networks are stochastic environments the throughput of any of an intersection's approaches will vary randomly. This results in noisy data. This could lead to difficulties in an agent determining whether it is more likely to be controlling the key intersection than any of its neighbors at any one point in time. In order to address this challenge Qoordination agents use the average throughput of each of their approaches that has been observed over the past cycle or a specified number of cycles. This averaging of the throughputs gives a much more steady result in the presence stochastically fluctuating traffic flow levels but also means that it takes a small number of cycles in order to detect a change in the main direction of traffic flow.

3.4.2 Phase Length Modifications

Each agent within the network can change phase lengths in accordance with whatever traffic control method they are implementing e.g. SAT. It is not required that all intersection agents within the transport network implement the same traffic control method. All non-key agents however must keep their cycle length the same as their neighbor intersection that they consider most likely to be controlling the key intersection. In this manner cycle lengths are decided upon by key intersection agents and these cycle lengths are cascaded through the network from intersection agents to their adjacent neighbors to whom the flow of traffic is moving. Progressive signal systems are not confined to single traffic corridors but can be dynamically branched out depending on the current flow of traffic. As all intersections along the traffic corridors now share the same cycle length their cycles are in sync such that coordination is possible.

3.4.3 Offset Modifications

The next step is to adjust the offset times such that a progressive signal system is established. As was discussed in the previous sub-sections Qoordination actions consist uniquely of offset adjustment actions. Qoordination rewards are based upon the minimization of queue lengths. The Qoordination agent's MLH utility function can calculate which offset adjustment actions are necessary for minimizing queue lengths. With minimizing queue lengths progressive signal systems emerge. Different offsets however are necessary in the event of a change of traffic flow direction. To take this into account Qoordination agents maintain a separate MLH utility function for each possible direction of traffic flow i.e. North, North West, West, South West, South, South East, East, and North East. The current direction of traffic flow is simply calculated based upon the approach with the highest throughput. For intersections with different layouts to those used in our evaluations the number of directions might be different. The number of directions can still however be calculated based upon an intersection's non-opposing approaches. The only MLH utility function that should be updated or queried during any time step is the one that is associated with the current traffic flow direction. Thus Qoordination agents learn how to establish progressive signal systems that can dynamically change direction or course. This approach of switching MLH according to dominant traffic flow direction is particularly suitable for transport networks that have a single dominant traffic flow direction at any one time. An agent only creates a MLH for a given direction once that direction has been detected as the main traffic flow direction. In our experiments each intersection only had one or two directions that were ever detected as the main direction so only one or two of a possible eight directional MLHs per agent were ever created. This approach is thus particularly scalable for intersection agents that experience few main directions of traffic flow. It is however not as suited to situations where there is no clear main traffic corridor i.e. where traffic flows equally in all directions. In this situation the all directional MLHs will be created and the MLHs being queried and

updated will continually switch. These continually switching MLHs will likely resemble each other more and more as time goes by. In this situation a more appropriate solution would be to have a single MLH. This would significantly cut down on memory requirements and would increase the agent's rate of learning.

3.4.4 Agent Abstraction

Most intersections do not depend on the offsets of all of their adjacent intersections to learn an accurate utility function. For example an intersection will tend to not be dependent on downstream adjacent intersections at most traffic flow levels, oversaturation being the exception. Qoordination agents can increase their learning rate by abstracting away unnecessary offset state variables from their MLH utility functions. Abstraction in MLH is achieved by simply adjusting the coarseness values. When it is known that a state variable is irrelevant then it can have its coarseness value set to a maximum. With the intersection agents used in our evaluation networks abstraction could be a trivial process of setting maximum coarseness values on all downstream intersections of the specific MLH direction. Thus an intersection agent's north-bound MLH utility function can rule out all neighbor offsets except for the adjacent intersection that is south of it. In our evaluation networks an agent can easily discover which adjacent intersection is to its north, south, east, or west, based upon their identification numbers and the total number of intersections in the network. In other networks this information is not as easy to obtain. In these situations a number of other processes can be followed in order to abstract irrelevant neighbor offsets. One approach is to search the MLH parameter space as has been described in section 3.3.9. Another similar approach is to cache or save to file a number of SARS tuples i.e. State, Action, Reward, State. A percentage of these tuples are fed into a number of initial MLH utility functions on initialization. This is a method of repopulating a MLH with information that it has previously learned so that it does not have to learn from scratch each time it is initialized. As each MLH only processes tuples that have resulted from its own previous experiences it cannot be comparable to transfer learning approaches that would allow agents to learn from the experiences of other agents (Taylor & Stone, 2009). In a transport network scenario like the one that we consider transfer learning is only potentially possible if an agent is fed tuples that have been experienced by another intersection agent that has identical characteristics e.g. number of approaches and number of neighboring agents, and that experiences close to identical patterns in traffic flow. Each initial MLH utility function is assigned a separate set of coarseness values. Thus one MLH may rule out one neighbor while another rules out another neighbor. The remainder of the cached tuples can be used to test the accuracy of the individual MLHs. The MLH that is most accurately able to estimate the immediate reward that is to be expected when performing a given action from a given state is the one that should be used. All others can be dropped. This process can even occur at regular intervals for each directional MLH utility function so as to ensure that no relevant neighbor offsets are being ruled

out. Ruling out of a relevant neighbor offset will have a negative effect on the agent's learning and action selection process. Initializing Qoordination agents' MLH utility functions in this way also ensures that they do not have to undergo a cold start.

3.5 Summary

In this chapter we have presented the requirements for a learning based approach to traffic intersection coordination. Based upon these requirements we have designed such an approach, namely Qoordination. This chapter describes the Qoordination design that meets the specified requirements. A detailed description was also given of Qoordination's novel MLH utility function.

Chapter 4

Implementation and Simulation-Based Evaluation Platform

In this chapter we describe in detail our simulation-based platform for evaluation of traffic control methods. This platform has a particular focus on the evaluation of the effects of coordination on traffic control methods. This evaluation platform is based upon the industry standard PTV VISSIM microscopic simulator (Fellendorf & Vortisch, 2010). We begin this chapter by giving a brief description of the VISSIM microscopic simulator and a description of how a custom written traffic control program can interface with it. We then detail the automated process of simulated network generation. We describe the platform's automated report generation functionality and then conclude with a description of the overall automated evaluation process.

This chapter also describes the implementation of a generic traffic control framework that was used for the development of Qoordination agents. This framework allows for flexibility to experiment with traffic controller agent types and their components. A description of our implementation of Qoordination agents and their integration into the simulation-based evaluation platform is then given.

4.1 Evaluation Platform

4.1.1 VISSIM Microscopic Simulator

PTV VISSIM is a leading industry standard microscopic traffic simulator that realistically models a variety of different road user types using scientifically sound models. Different road user types include

car, heavy goods vehicle, bus, tram, bike, and even pedestrian. VISSIM has the advantage of ease of configuring a simulated transportation network model. A user can import an image obtained from a mapping service such as Google Maps, and then overlay this image with links and connectors representing the road network. Other elements of the road network can then be added to this road infrastructure including but not limited to: speed restrictions, priority rules, stop and yield signs for uncontrolled intersections, routing decisions, vehicle inputs, scheduled public transport lines, signal controllers, and induction loops. These elements provide considerable flexibility in traffic flow levels, traffic patterns, and intersection control. A number of signal controller methods are also included with VISSIM. The main two signal controller methods are Vissig and VAP. Vissig implements pre-timed cyclic signal control (see section 2.3.2.1) while VAP implements actuated signal control (see section 2.3.2.2). An essential aspect of VISSIM that enables us to use it as a base for our evaluation platform in this thesis is its programming interfaces that provide seamless integration with custom written signal control logic. The first programming interface is the DriverModel.DLL interface which allows for the implementation of custom car-following and lane changing models that overwrite the standard driving behavior. In this thesis as we do not model vehicular agents so we are not as interested in this interface as we are in the second interface, namely the SignalControl.DLL interface. The SignalControl.DLL interface allows for integration of user-defined signal control logic, as illustrated below.

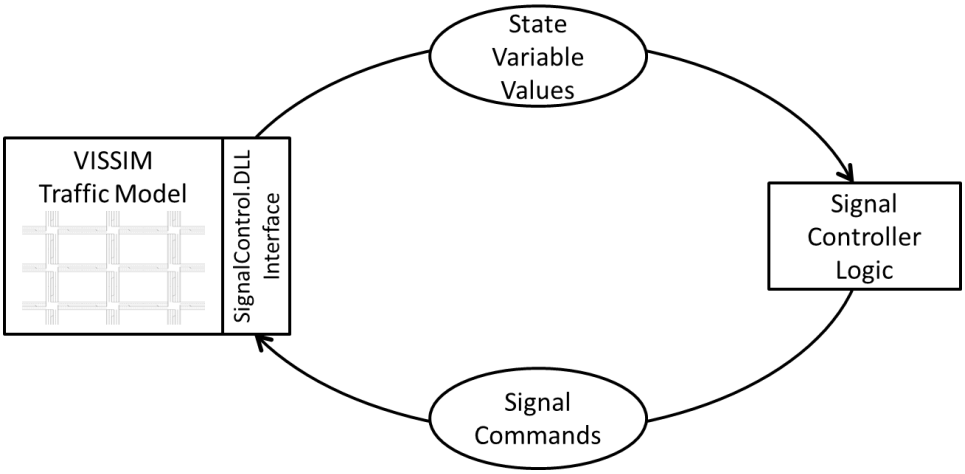

Figure 39 The VISSIM signal controller programming interface

It is using this interface that we integrate our traffic control methods into the VISSIM simulations. VISSIM provides comprehensive analysis options, reinforcing its suitability as a base for our evaluation platform. VISSIM also provides a COM interface for running simulations programmatically. This is of high importance for automation of the evaluation process. We use VISSIM version 5.20 for our evaluation platform.

4.1.2 Network Generation

In evaluating the effects of coordination on a traffic control method we begin by evaluating its performance on a simulated model of a single intersection. We then perform the same evaluation but on a 2x2 intersection grid network, then on a 3x3 grid, then on a 4x4 grid, and thus the grid networks continue to incrementally grow for each subsequent evaluation step up to a specified maximum grid size e.g. a 100x100 grid network. The grid network is a common transport network topology in practice e.g. in cities such as Manhattan in the USA. The grid network is also a configuration that is commonly used in evaluating traffic control systems (see examples in section 2.3.3). Although setting up a simulated network in VISSIM is not difficult the process of setting up a grid network consisting of 1,000 intersections would be tedious and error prone. For this reason this evaluation platform automates the generation of simulated traffic networks.

VISSIM network models are stored as human-readable ASCII .inp files. The parameters for each element within a simulated network are thus contained within its .inp file in an intuitive, well organized, categorical fashion. In order to automate the process of network generation we must programmatically create valid .inp files. The first step to doing this is to model a single ideal intersection in VISSIM that has all necessary approaches, lanes, priority rules, communication channels, detectors, and traffic lights. The next step is to programmatically clone this intersection a number of times and reposition and connect the intersection instances in a specified order to form the desired transport network. The specifications for the network size and topology, traffic flow levels, traffic patterns, etc. are read from a parameter file.

In the remainder of this section we give a detailed description of this automated network generation process.

4.1.2.1 Single Intersection

As mentioned above, the first step in the process of automated simulated transport network generation is to model a single ideal intersection in VISSIM that has all necessary approaches, lanes, priority rules, communication channels, detectors, and traffic lights. As shown in Figure 40, the intersection layout that we modeled is the same as is illustrated in Figure 19.

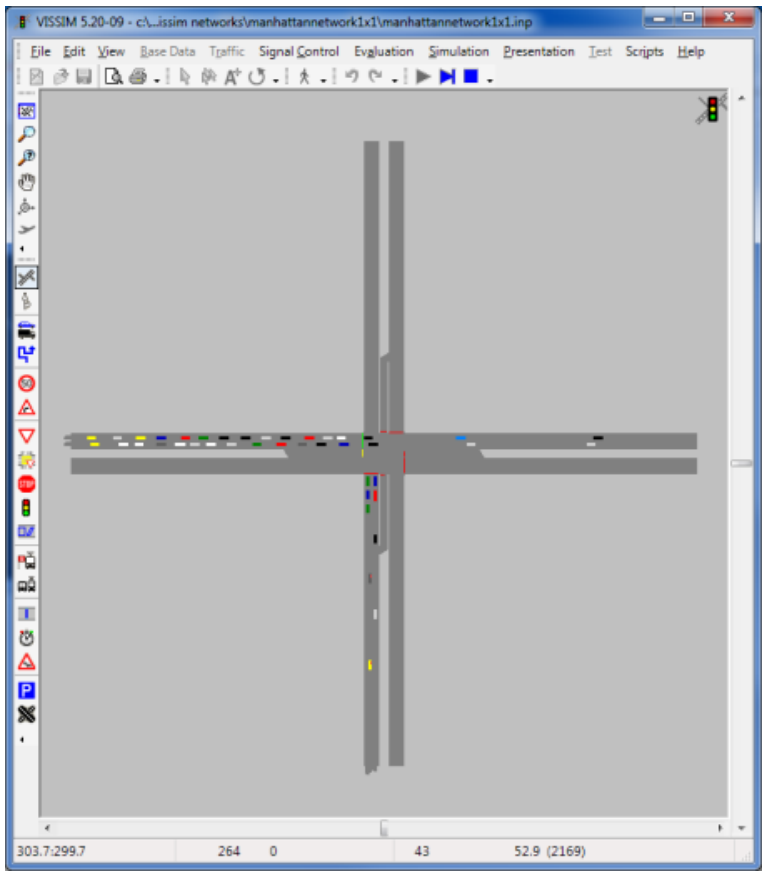

Figure 40 Single simulated intersection

In this model the vehicles drive on the left side of the road. The intersection has four arms, each one having three inbound lanes and two outbound lanes. The inbound lanes allow for vehicles to take left and right turns as well as to continue straight through the intersection. The rightmost lane on each approach allows for dedicated right turns and no straight through movements. The priority rules are set up so that vehicles turning right give way to vehicles going straight through the intersection. Vehicles also do not enter the intersection if their exit from the intersection is not clear, thus avoiding unnecessary congestion and gridlock. With these priority rules in place, as well as various others, vehicles can safely navigate through the intersection even when it is uncontrolled. Traffic lights are placed on each lane of each approach to the intersection. Detectors are also placed on each lane of each approach at distances of 5 meters and 30 meters from the intersection, as seen in Figure 41.

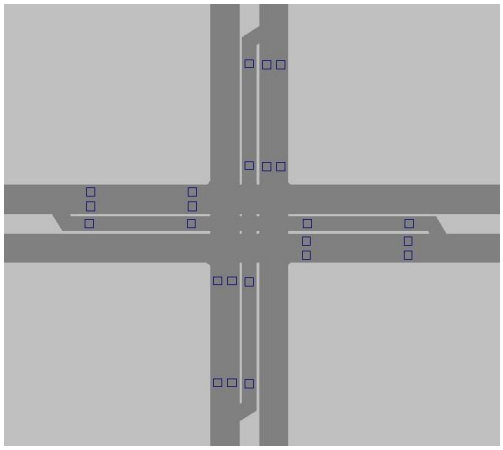

Figure 41 Detector placement at intersection

These detectors are named according to a convention that allows them to be programmatically identified by approach, lane, and distance to intersection. Communication channels are then set up in such a way that the intersection has one single output channel and one input channel for each approach. This is to eventually allow the intersection agent to receive information from each of its neighbor intersection agents individually and to send information to them as a group. The decision to send information to them as a group comes from the fact that when a vehicle exits the intersection we do not know to which of the neighboring intersections it is actually going, e.g. we do not know whether a vehicle in the left lane of an approach will be turning left or going straight through the intersection. We do know however which neighbor intersections the arriving vehicles are coming from, hence the individual input communication channels. These input channels are also named according to conventions to allow for programmatic management. The final step in setting up the single intersection is to put the traffic lights into a signal group and to set the name of the DLL interface file that this signal group is to use. We set the interface to SignalControl.DLL, which we can then modify to implement our traffic control logic (see section 4.2.1).

4.1.2.2 Multiple Intersections

Now that a single intersection has been modeled we can programmatically clone it a number of times and relocate the clones within the model by adjusting variable values within the .inp file. The next step is to connect the outgoing lanes to the incoming lanes of adjacent intersection instances. This is facilitated by organized naming conventions of intersections and roads. The distances between intersections are defined by adjustable variables, allowing for a greater variety in network topology.

On each lane that leads into the transportation network we set speed limits and vehicle inputs. Speed limits of 50 kilometers per hour were set as is common in urban areas. A vehicle input depicts how many vehicles are to enter the network from the lane on which it is set. The input volumes are controlled programmatically by manipulating the .inp file and can even vary throughout the duration of a simulation

evaluation. We can also easily stop all traffic from coming from a specified direction, enabling us to easily simplify the traffic flow patterns for purposes of development and evaluation.

Routing decisions are then set from each incoming road to each outgoing road. Only one route is plotted for each entrance-exit pair as opposed to plotting an exhaustive set of routes. Traffic flows within the transport network are thus specified using Origin-Destination (OD) pairs implemented as routing decisions as opposed to being specified using turning probabilities. Main traffic flows can thus be set by increasing the vehicle input for the desired OD pair. As the plotting of these routes is done programmatically it becomes easy to state whether left or right turns are allowed within the network, which again enables us to easily simplify the traffic flow patterns for purposes of development and evaluation. Figure 42 illustrates one of the automatically generated routes through a multi-intersection grid network that has various distances between intersection columns and rows.

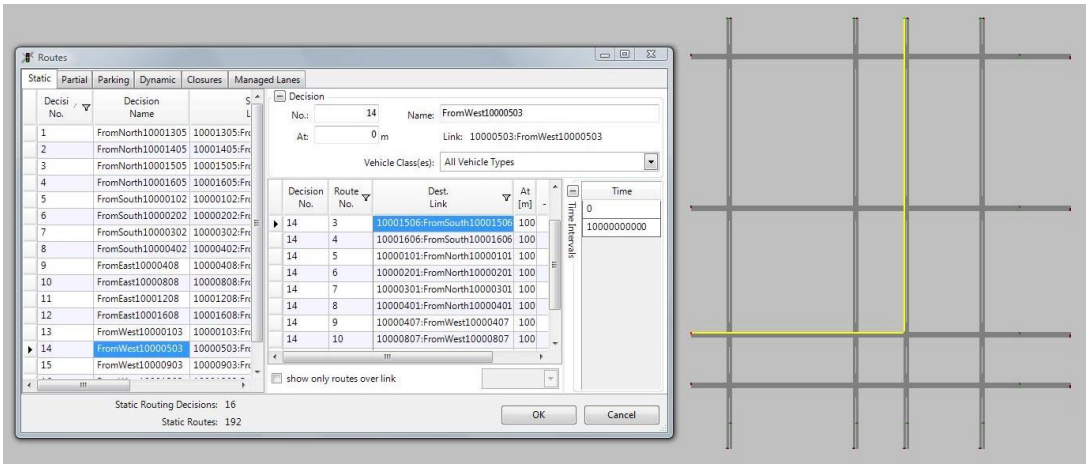

Figure 42 Automatically generated route through a simulated grid network

4.1.2.3 Parameter File

Having now the ability to programmatically generate a simulated transport model we set up an ASCII parameter file to define the appropriate network properties. This file contains the following parameters.

- Simulation duration
- Number of times to run the simulation
- Distances between each row and column of intersections within the grid network
- The traffic flow levels from each direction
- The ratio of vehicles per source direction that will be turning left, right, or going straight through the network
- The range of iterative network size increments

Of these parameters it is perhaps the last one that requires an explanation, which will be done in the following sub-section.

4.1.2.4 Incremental Network Size Increase

We have now seen that the specifications of the simulated transport network to be generated can be set in the parameter file. If we wish to generate a number of networks, each one being incrementally bigger than the last, then a separate parameter file would need to be defined for each one. This approach can be tedious and error prone if the number of networks is large. To overcome this challenge an iterative network size increment range can be set in the parameter file. This sets the minimum network size, the maximum network size, and the increment size. For example, if a minimum network size of 1 is set and a maximum network size of 10, with an increment size of 2, then the following network sizes will be generated: 1x1, 3x3, 5x5, 7x7, and 9x9. Besides of network size the other parameters specified remain the same for each network. Below is shown an example of a 5x5 and a 20x20 simulated transport network generated using the same parameter file.

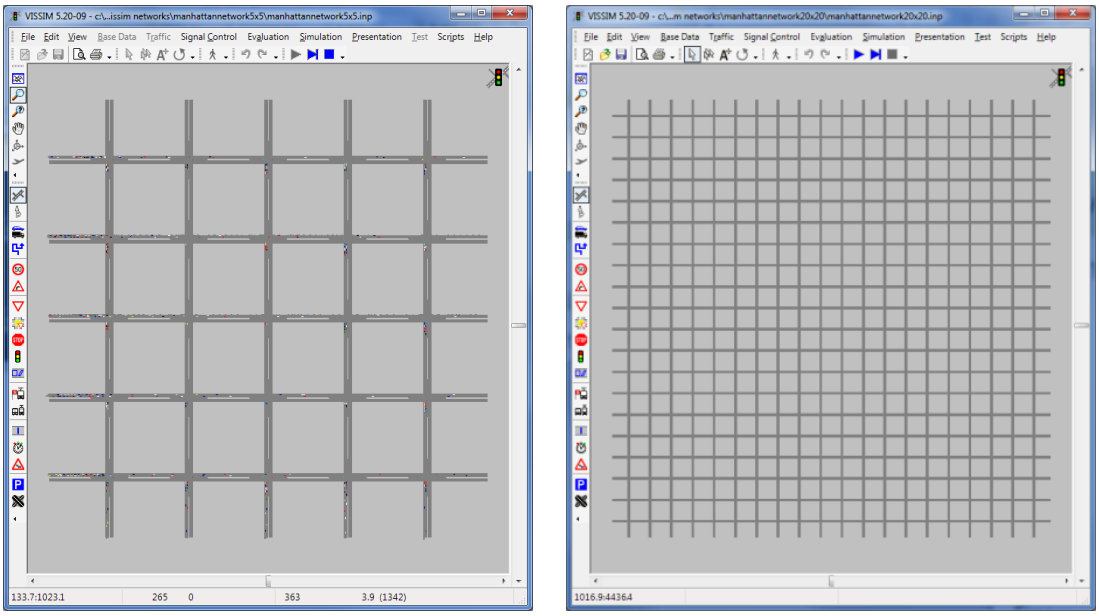

Figure 43 Automatically generated 5x5 and 20x20 simulated grid networks

4.1.3 Report Generation

As mentioned in section 4.1.1, VISSIM provides comprehensive analysis options allowing us to evaluate the performance of a traffic control method over the course of a simulation. The main evaluation type that VISSIM provides that we are interested in is the vehicle record. A number of parameters can be set in the vehicle record, including simulation time, vehicle number, route number, distance traveled in

network, queue time, number of stops, and vehicle speed. These variables give us valuable information that we cannot gather from the detectors alone, whereas the traffic controller logic that we implement in the DLL file only has available to it the information that can be gathered from the detectors. VISSIM exports the vehicle record to a .fzp file. This file contains thousands of entries, one for each vehicle at each time step e.g. each second, of the simulation. Our evaluation platform compiles these results automatically to generate a report for each simulation that shows: accurate vehicle speeds, vehicle throughput, traffic volume, average queue time, average number of stops, etc. If the simulation is run numerous times for a single network size then the results of all of these simulation runs are averaged out, allowing us to see performance patterns clearer by reducing erratic results that are associated with such a stochastic environment. Stochasticity of the environment is represented in VISSIM by fluctuating the rate at which vehicles are introduced into the network through the vehicle input nodes. Although the average over the course of an hour may stay fairly consistent the first ten minutes of that hour may see more vehicles introduced than the second ten minutes of the hour. The randomness of the rate of introduction of vehicles into the network through the vehicle input nodes is affected by a seed variable. Each time a simulation is run we increment this seed variable so that the results of a sequence of simulation runs will all be different. It is because the simulation results vary significantly that we average them over a number of simulation runs. These evaluation variables give us a good understanding of how the traffic control logic performs per network size, yet at times we will wish to analyze performance at the level of the intersections. To do this we print out variables for the individual intersections such as waiting time, queue length, throughput, and reward. At each time step of the simulation a “calculate” callback method is called in the SignalControl.DLL code with the signal controller number of an intersection given as a parameter. This callback method gives the intersections controlling agent the opportunity to call any methods necessary to update its state and perform any actions. This also gives us the opportunity to print out these evaluation variable values. These variables are also averaged across intersections and simulations and put into a report file. It is also however possible to see how a specific intersection is performing. This can be important when analyzing the different performances of intersections based upon their location in the network, or along a specific traffic corridor. Individual route performance can also be seen in the evaluation report.

When a report file has been generated the results are programmatically entered into a Microsoft Excel spreadsheet using Excel’s COM interface. Separate graphs are generated within this document to allow for easy comparison of traffic control algorithms in different sized networks.

4.1.4 Automated Evaluation

Our evaluation platform automates the evaluation process of different traffic control algorithms. A parameter file is defined in which is stored the evaluation specific settings. For each algorithm to be

tested this file firstly states the range of iterative network size increments. It then states the duration of the simulation and the number of times the simulation is to be run. A separate number of training simulations can be run beforehand for learning algorithms, which allow agents to build up their models and utility functions but does not include the results in the final report. The parameter file then states whether or not the evaluation should be done on a per route or a per intersection basis or if it should simply be averaged over the entire network. The former have an obvious temporal cost, particularly in larger networks. The granularity of the resulting graphs can also be set so as to allow for smoothing or a more detailed view. Each traffic control algorithm can be evaluated using a number of different traffic flow levels, or even using a varying traffic flow level. The settings for these traffic flow levels are all set in the parameter file. The final entries in the parameter file state the traffic control algorithms to be evaluated. The output from the automated evaluation is an Excel spreadsheet diagrammatically depicting each algorithm's performance.

4.2 Traffic Control Framework

The traffic control framework developed as part of this thesis offers flexibility and structure in experimenting with different RL and traffic control based design elements such as agent type, learning algorithm, function approximator, environment representation, reward model, state space, and action space. Within this framework the agent and the traffic controller components are decoupled. The reason for this decoupling is that we wanted to reserve the ability to integrate our Agent implementation with other environments that are unrelated to traffic control with a minimum of required alterations to the Agent code. For this reason the agent and traffic controller components have been implemented quite independently. This has led to some duplication of class names between components that could have been avoided had a more tightly integrated design been considered. A high level class diagram of the traffic control framework is presented in Figure 44.

We now describe how this framework integrates into the evaluation platform. The remainder of this sub-section then describes the component pieces of the framework.

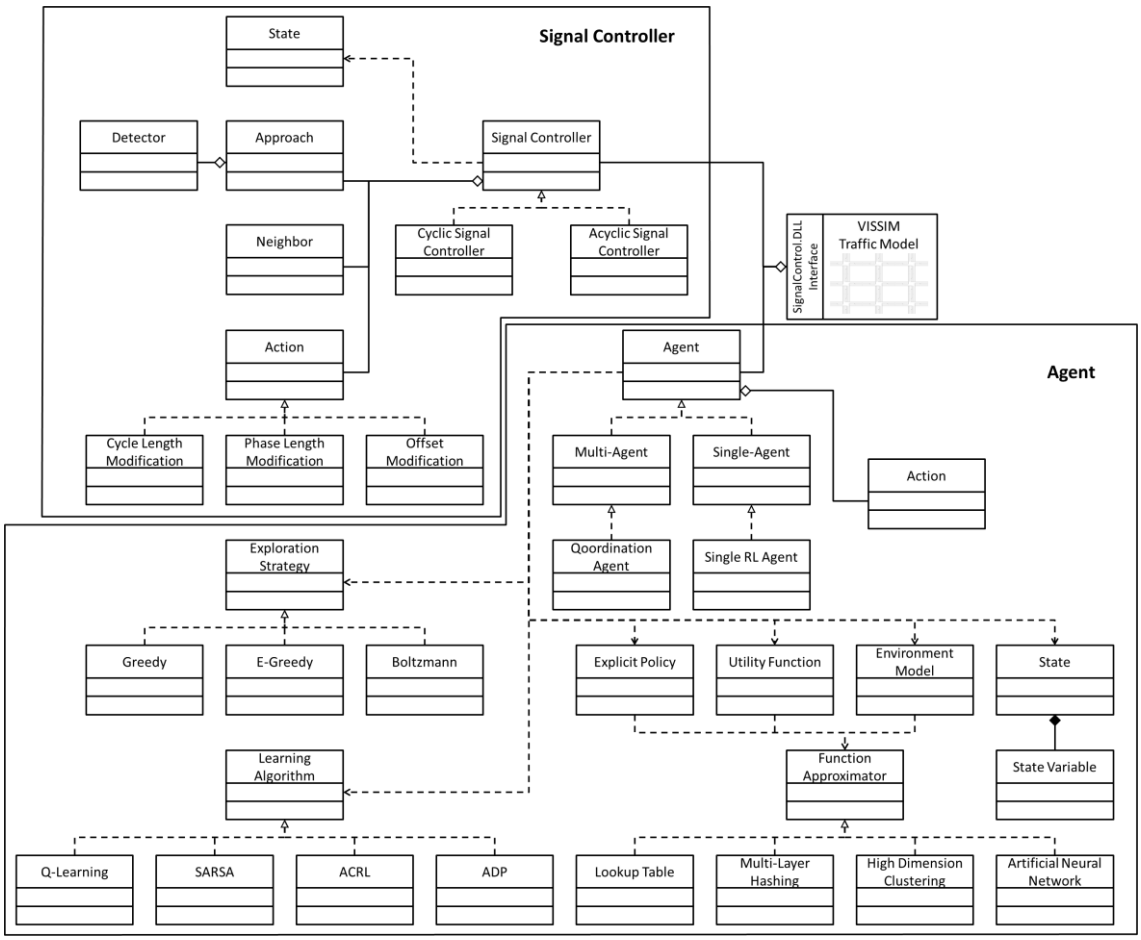

Figure 44 Traffic control framework high-level class diagram

4.2.1 Integration with Evaluation Platform

As illustrated in Figure 44 the VISSIM microscopic traffic simulator interacts with the signal controller logic via the SignalControl.DLL interface. The DLL interface is coded in the C++ programming language and provides a set of callback methods that are called for each traffic controller during each time step. Custom functionality can then be placed inside these callback methods. This functionality can call upon some of the functions provided by the API to discover the state of the traffic network e.g. to discover the number of vehicles that have passed over a given traffic detector within the past time step, or to discover the current state of a given signal group (e.g. green, amber, or red). The functions provided by the API also allow for the state of the signal controllers to be updated. The updated states of the signal controllers are then passed back to the VISSIM simulation. We include the following list of variables and methods to illustrate the functionality that is provided by the VISSIM API.

```
/*========================= General functions ============================*/
double Sim_Time(void); // Returns the simulation time at the end of the current
time step in VISSIM in seconds.
char *Sim_Time_of_Day(void); // Returns the simulation time at the end of the
current time step in VISSIM as string in the form hh:mm:ss.s.
long Sim_Start_Date(void); // Returns the start date of the simulation as defined
by the user in the VISSIM Simulation Parameters in the format YYYYMMDD.


/*====================== Signal controller functions =====================*/
double  SC_CycleSecond (unsigned long sc_no); // Returns the current cycle second
[in seconds, >0 <=SC_CycleTime()] of controller no. <sc_no>.

double   SC_CycleTime (unsigned long sc_no); // Returns the cycle time (in
seconds) of controller no. <sc_no> (as defined in VISSIM).
long  SC_ReadInputChannel (unsigned long sc_no, long ch_no); // Returns the last
value sent from the controller output channel connected to input channel <ch_no>
of controller no. <sc_no>.
int  SC_SetOutputChannel (unsigned long sc_no, long ch_no, long value); // Sets
the output channel <ch_no> of controller no. <sc_no> to <value>.

/*========================= Signal group functions ========================*/
// State definitions
#define SG_STATE_UNDEFINED           0
#define SG_STATE_GREEN               1
#define SG_STATE_RED                 2
#define SG_STATE_OFF                 4
#define SG_STATE_RED_AMBER           7
#define SG_STATE_AMBER               8
#define SG_STATE_AMBER_FLASHING      9
#define SG_STATE_RED_FLASHING       10
#define SG_STATE_GREEN_FLASHING     11
#define SG_STATE_RED_GREEN_FLASHING 12
#define SG_STATE_GREEN_AMBER        13

sg_iterator  SignalGroups  (void);  //  Returns  an  iterator  containing  all
SignalGroup objects.
int SGIdFromName (unsigned long sc_no, const std::string & name); // Gets the id
of the signal group with the given name.
std::string SG_Name(unsigned long sc_no, unsigned long det_no); // Gets the name
of the signal group with the given id.
int   SG_SetState (unsigned long sc_no, unsigned long sg_no, int state, int
transition); // Set signal group no. <sg_no> of controller no. <sc_no> to <state>
(see state definitions above).
int  SG_CurrentState (unsigned long sc_no, unsigned long sg_no); // Returns the
current state of signal group no. <sg_no> of controller no. <sc_no> (see state
definitions above).
double  SG_CurrentDuration (unsigned long sc_no, unsigned long sg_no); // Returns
the elapsed time (in seconds) since signal group no. <sg_no> of controller no.
<sc_no> was set to its current state.
double  SG_MinimumRed (unsigned long sc_no, unsigned long sg_no); // Returns the
minimum red time of signal group no. <sg_no> of controller no. <sc_no> (in
seconds, as defined in VISSIM).
double  SG_MinimumGreen (unsigned long sc_no, unsigned long sg_no); // Returns
the minimum green time of signal group no. <sg_no> of controller no. <sc_no> (in
```

```
seconds, as defined in VISSIM).
double  SG_AmberPeriod (unsigned long sc_no, unsigned long sg_no); // Returns the
amber time of signal group no. <sg_no> of controller no. <sc_no> (in seconds, as
defined in VISSIM).
double  SG_RedAmberPeriod (unsigned long sc_no, unsigned long sg_no); // Returns
the red/amber time of signal group no. <sg_no> of controller no. <sc_no> (in
seconds, as defined in VISSIM).

/*========================== Detector functions ==========================*/
detector_iterator  Detectors  (void);  //  Returns  an  iterator  containing  all
detector objects.
std::string Det_Name (unsigned long sc_no, unsigned long det_no); // Gets the
name of the detector with the given id.
int  Det_Detection (unsigned long sc_no, unsigned long det_no); // Returns 1 if
there is or was a vehicle on detector <det_no> of controller no. <sc_no> since
the previous controller time step, else 0.
int  Det_Presence (unsigned long sc_no, unsigned long det_no); // Returns 1 if
there is a vehicle on detector <det_no> of controller no. <sc_no> at the end of
the current simulation time step, else 0.

int  Det_FrontEnds (unsigned long sc_no, unsigned long det_no); // Returns the
number of detected vehicle front ends on detector <det_no> of controller no.
<sc_no> since the previous controller time step. (If a vehicle moves onto the
detector while another one is still there, no new front end will be detected!)
int  Det_RearEnds (unsigned long sc_no, unsigned long det_no); // Returns the
number of detected vehicle rear ends on detector <det_no> of controller no.
<sc_no> since the previous controller time step. (If a vehicle leaves the
detector while another one is already on the detector, no new rear end will be
detected!)
double  Det_FrontEndTime (unsigned long sc_no, unsigned long det_no, int k); //
Returns the time in s between the start of the current controller time step and
the detection of the <k>-th vehicle front end during this controller time step.
[1 <= k <= Det_FrontEnds (sc_no, det_no)]
double  Det_RearEndTime (unsigned long sc_no, unsigned long det_no, int k); //
Returns the time in s between the start of the current controller time step and
the detection of the <k>-th vehicle rear end during this controller time step. [1
<= k <= Det_RearEnds (sc_no, det_no)]
double  Det_OccupancyTime (unsigned long sc_no, unsigned long det_no); // Returns
the current occupancy time on detector <det_no> of controller no. <sc_no>: 0.0 if
no vehicle is present at the end of the current simulation time step, else the
time elapsed since its arrival in s.
double Det_OccupancyRate (unsigned long sc_no, unsigned long det_no); // Returns
the current occupancy rate of detector <det_no> of controller no. <sc_no> in %
[0.0..100.0], i.e. the occupancy percentage since the last controller time step.
double Det_OccupancyRateSmoothed (unsigned long sc_no, unsigned long det_no); //
Returns the exponentially smoothed occupancy rate of detector <det_no> of
controller no. <sc_no> in % [0.0..100.0], i.e. the occupancy percentage since the
last controller time step, smoothed exponentially with the smoothing factors
entered for this detector in VISSIM.
double  Det_GapTime (unsigned long sc_no, unsigned long det_no); // Returns the
exponentially smoothed occupancy rate of detector <det_no> of controller no.
<sc_no> in % [0.0..100.0], i.e. the occupancy percentage since the last
controller time step, smoothed exponentially with the smoothing factors entered
for this detector in VISSIM.
```

```
double  Det_VehSpeed (unsigned long sc_no, unsigned long det_no); // Returns the current vehicle speed on detector <det_no> of controller no. <sc_no>: 0.0 if no vehicle was detected since the last controller time step, else the speed of the last vehicle detected in m/s.
double  Det_VehLength (unsigned long sc_no, unsigned long det_no); // Returns the current vehicle length on detector <det_no> of controller no. <sc_no>: 0.0 if no vehicle was detected since the last controller time step, else the length of the last vehicle detected in m.
int DetIdFromName (unsigned long sc_no, const std::string & name); // Gets the id of the detector with the given name.
short  Det_Type  (unsigned  long  sc_no,  unsigned  long  det_no);  //  Returns  the detector type. With use of type it is possible to differ between detectors for pedestrians and detectors for the traffic.
```

4.2.2 Signal Controller

Each Signal Controller object represents a separate intersection signal controller within the simulated VISSIM model. These objects have access to the current sensor readings and methods for performing actions on their associated VISSIM intersections. This traffic controller aspect of the framework is decoupled from the agent based learning aspect of the framework.

4.2.2.1 Initialization

When the Signal Controller object is first initialized it instantiates the appropriate State, Neighbor, Approach, and Detector objects. Signal Controllers are aware of which intersections in the network are adjacent to them as the communications channels are automatically set up between adjacent intersections when the simulated network model is generated. Alternatively, Signal Controllers are made aware of which intersections are adjacent to them in the network by directly accessing the Signal Controllers array. The approaches leading in to each intersection are given by VISSIM under the name of Signal Groups. Using signal group identifiers the Signal Controller can discover the current state of the traffic lights on each individual approach. It can also use these identifiers in changing the traffic lights on any approach. Using the Approach identifier the Signal Controller can also discover any detectors located on the approach. Two detectors are located on each approach of each intersection in each of our generated VISSIM networks. These detectors are located at 5 and 30 meters from the intersection. Using this information the Signal Controller can instantiate Detector objects that are assigned appropriate detector identification numbers. These detector identification numbers are used to discover how many vehicles have passed over the associated detector. Other methods that can be called using the approach identifier and the detector identifier are given in the list of methods on the previous page. Using the combination of the two detectors the Signal Controller can calculate the vehicle queue length on each approach.

Once the appropriate Approach and Detector objects have been instantiated the Signal Controller initializes the phases. Phases are implemented as arrays of Approach references. All approaches whose

traffic lights are to receive green time are grouped together in one of these arrays. As references to the Approaches are used a single approach can be included in multiple phases, allowing for overlapping phases such as those shown in Figure 21. Our implementation defines a six phase cycle. This is to allow for dedicated right turns so as to decrease the risk of congestion in case of heavy traffic for pre-timed control. Once the phases have been established then the appropriate Action objects can be instantiated. This generic framework allows for the development of multiple types of Signal Controller logic. For example, our Qoordinated agents only require three actions i.e. increasing offset, decreasing offset, and performing no action whereas the Q-Learning based agents that we use in our evaluations require three actions for each phase i.e. a phase increase modification, a phase decrease modification action, and an action where no modifications are made to the specified phase length.

4.2.2.2 Update

When an update message is received from the simulated signal controller each Detector object updates its variables in accordance with the information contained within the update message. The Approach objects are then able to observe how many vehicles have arrived in the queue i.e. how many have passed the detector at 30 meters within the past time step, and how many have left the queue i.e. how many have passed the detector at 5 meters within the past time step. Using this information the Approach objects can not only maintain accurate queue length information but also waiting time information. Each Approach object maintains a history of these variables as well as a history of the traffic lights state as obtained from the update messages. Using these histories the Approach object is able to calculate their averages over the course of a cycle length, or even over a given number of cycle lengths. These averaged values can then be fed into the appropriate Agent State Variables.

The Signal Controller object keeps track of the cycle time expired so far and also the current phase time expired so far. When a phase is nearing its end (about 4 seconds before it ends) its approach's traffic lights are set to amber. For the last second of the phase the traffic lights are set to red. This concludes the phase's change and clearance intervals. On the first time step of the next phase the traffic lights of the approaches that are included in that next phase are turned to green. If an approach is included in two consecutive phases then it stays green during its change and clearance intervals. If a phase has a phase length of zero then its green, change, and clearance intervals are skipped.

4.2.2.3 Action Performance

Cyclic Signal Controllers can perform actions at the start of a cycle. Phase length modification actions are performed simply by reducing or increasing the length variable value for the specified phase by a specified amount. Phase length modifications are a fixed 6 second duration. Thus agents that adjust phase lengths cannot adjust them by any more or less than 6 seconds at a time. Offset modification actions are

performed by reducing or increasing the cycle length for the duration of one full cycle. This temporary adjustment is spread out evenly among each phase so that their times are uniformly increased or decreased just for the duration of that cycle. An agent can thus only modify its offset to all of its adjacent neighbors at once and cannot modify its offset to any one of them individually. Offset adjustment sizes are a fixed duration that is a multiple of the intersection's number of phases. In our evaluation scenarios each intersection has six phases and the intersection agent can modify its offset values by 12 seconds. Having six phases allows for dedicated right turns in all directions so as to decrease the risk of congestion in case of heavy traffic for pre-timed control.

4.2.3 Agent

The Agent object represents the learning aspect of the framework and is decoupled from the traffic controller based aspect of the framework. Agents within the traffic control framework can have the following components: exploration strategy, learning algorithm, utility function, policy, model, state, and a set of actions.

4.2.3.1 State and Actions

The Agent State and Actions represent the State and Action objects maintained by the Agent's corresponding Signal Controller. Each Action simply contains a value and an identification that links it to its Signal Controller counterpart. The action value represents the increase or decrease in the specified offset or phase length. Each State Variable also contains a value and an identification that links it to an element of the Signal Controller's State, such as average queue lengths, waiting times, or phase lengths. These values are obtained from the corresponding Signal Controller State after an update has occurred. Neither Actions nor State Variables have any concept of the variables that they represent and simply act as their proxies within the learning section of the framework. This allows for decoupling between agent and traffic controller.

4.2.3.2 Abstract Classes

Abstract Exploration Strategy, Learning Algorithm, and Function Approximator elements facilitate experimentation with different potential design components. Different Exploration Strategies such as Greedy, E-Greedy, and Boltzmann, have been implemented (see section 3.2.9), as well as combinations of these. During the action selection process the Agent calls a getNextAction method on the Exploration Strategy, which decides whether to perform an explorative action or whether to return the exploitative action obtained by calling the Policy's getNextAction method. The agent's utility function, policy, and environment model are all represented using a function approximator. Where an implicit policy is desired the Policy object's getNextAction method simply consults the Utility Function and returns the Action

with the highest utility. Not every Learning Algorithm type will make use of the Model e.g. Q-Learning. These are model free algorithms. Implemented Function Approximators include: Lookup Table, High Dimensional Clustering, Artificial Neural Network, and Multi-Layer Hashing. Learning algorithms that have been implemented include Q-Learning, SARSA, ACRL, and ADP. The agent itself can be either a Multi-Agent Agent or a Single-Agent Agent. Single Agents do not communicate with any of their neighbor intersection agents and remain unaware of the states, actions, and rewards of their neighbors, to the point of being unaware that there are even any other agents within the system.

4.2.3.3 Initialization

During the initialization process a new Agent object is instantiated for each Signal Controller. A parameter file is set up that defines the different design elements that are to be incorporated into the instantiated Agent. Initialization from a parameter file is done so as to facilitate automated evaluation of different agent types. If a simple pre-timed or actuated agent is being instantiated i.e. no learning element is involved, then a shell of an Agent object is created that performs no processing and whose getNextAction method consistently returns the "perform no action" action. When a learning Agent is being instantiated it reads a file containing saved experiences from previously run simulations that that particular agent participated in. Each entry contains a State, the Action that had been selected, the reward that was received after the action had been performed, and the resulting state. This initialization allows the Agent to avoid a cold start and provides the Agent with an established function approximator at the time of its instantiation. As the initialization files are specific to not only the signal controller but also the network size, learning algorithm, and traffic flow levels, a number of initialization files are maintained to significantly decrease development and testing times. This is also helpful for automated evaluation.

4.2.3.4 Update

The simulated signal controller sends an update at every time step of the simulation. This is roughly once per second. This update information is used to update the Signal Controller object. This update rate is too high for a learning agent as it does not give enough time to evaluate the effects of the performed actions. The agent must thus wait a number of cycles so that the effects of the performed actions become apparent. The number of cycles that we have found appropriate is two. The reward is calculated from data averaged over the past cycle.

The Agent thus receives an update at the first time step of every second cycle. This update begins by updating the Agent's State Variables from the values within the Signal Controller's State. What happens next in the update is specific to whichever learning algorithm is being followed. The learning algorithms generally update the Policy, Utility Function, Model, and perhaps even the Exploration Strategy at this point, although many algorithms simply update the Utility Function. We will give a detailed description

of Qoordination agent learning in the next section. Once the update process has been completed the process of action selection begins. This process begins when the Agent calls a getNextAction method on the Exploration Strategy, which decides whether to perform an explorative action or whether to return the exploitative action obtained by calling the Policy's getNextAction method. The Policy can either suggest the action that has the highest utility as defined by the utility function (implicit policy) or it can perform processing of its own in a search to become the optimal policy (explicit policy). Our Qoordination agent implementation uses the implicit policy functionality based upon a MLH utility function. MLH implementation will be described in greater detail in section 4.3. Once the action to be performed has been chosen it is returned to Signal Controller object which performs the action via SignalControl.DLL methods. The state, action, reward, state tuple is then saved to file.

4.3 Qoordination Agents

Qoordination Agents are Multi-Agent Agents that implement the Q-Learning Algorithm, a Multi-Layer Hashing Utility Function, no explicit Policy, no Model, and an Exploration Strategy whose functionality has been described in section 3.2.9. Qoordination agents contain State Variables that represent the intersection's offset to each adjacent intersection. Only three Action objects are required for a Qoordination agent. These represent actions for increasing or decreasing the intersection's offset to its neighbor intersections, and one Action to represent the absence of an action i.e. the offsets are kept steady.

Qoordination agents use an MLH function approximator to represent their Q-Learning utility function. The MLH function approximator implementation is illustrated using the class diagram that can be seen in Figure 45,

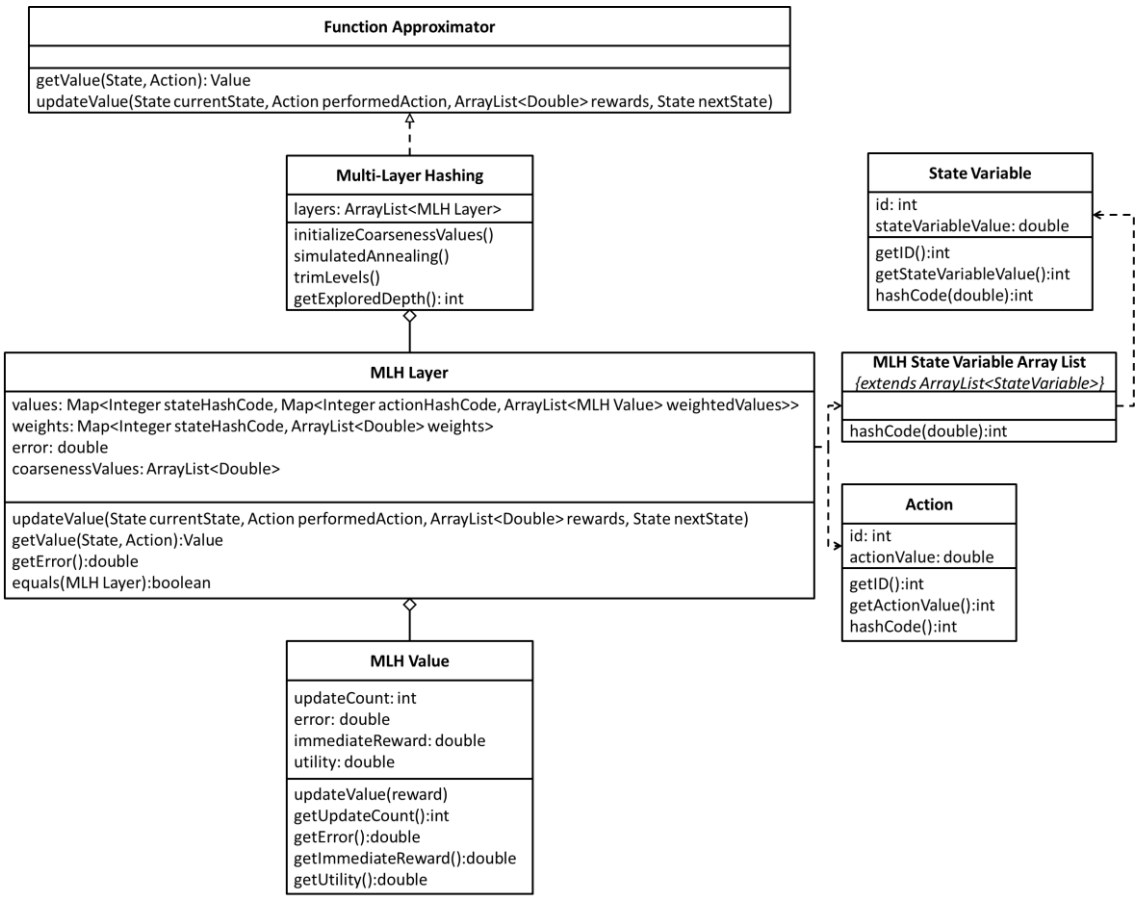

Figure 45 MLH class diagram

Qoordination agents maintain a MLH utility function not only for themselves but also for each of their neighbors. Thus a Qoordination agent is aware of how its own actions affect not only itself but also each of its neighbors individually. Further, Qoordination agents maintain separate MLH utility functions for each possible direction of traffic flow. The reason why this approach was taken as opposed to storing all of this information in the same MLH utility function was that during the process of abstraction different traffic flow directions require the abstraction of different neighbor offsets. For example, when traffic is flowing north the offset of the intersection to the north is typically abstracted and when the traffic is flowing south the offset of the intersection to the south is typically abstracted.

4.3.1 Initialization

This function approximator is initialized by first having a number of MLHs that are generated with different sets of coarseness values. In larger systems random sets of coarseness values can be used, followed by a process of simulated annealing as described in section 3.3.9. As the number of adjacent neighbor intersections is quite small an exhaustive set of coarseness value combinations is used. Each of

these utility functions are initialized using a percentage (in our case using 50%) of the SARS tuples i.e. State, Action, Reward, State, that have been saved to file during previous simulations. The remaining percentage of these tuples are used to test the accuracy of each utility function. The utility function that is able to most accurately estimate the immediate reward to be received for performing a given action from a given state is kept while all other utility functions are dropped.

Qoordination agents require a utility function for both themselves and for each of their neighbors. A separate MLH is thus created for each of these. Using these a Qoordination agent can be aware of the effects of its actions on both itself and on each of its adjacent neighbors individually. Further, each agent is to maintain a separate set of MLH utility functions for each direction of traffic flow. As Qoordination agents only update and access utility functions for the current traffic flow direction they typically do not create sets of utility functions for directions in which the traffic does not flow. In our experiments Qoordination agents typically only have to create sets of MLH utility functions for one or two directions as only one or two main traffic flow directions are investigated at a time. This solution is particularly scalable in situations such as those presented in our experiments i.e. where the number of neighbor agents is quite small (4) and where there are only one or two main directions of traffic flow per intersection. This solution is not as scalable in situations where there are higher numbers of neighboring agents and where the number of main directions of traffic flow are high e.g. where traffic flows equally in all directions.

When the Qoordination agents have been initialized for the first time they must perform each of their three possible actions a number of times without any other relevant agents in the network performing any actions. This is to allow the agents to know the effects of their actions alone, as opposed to their actions when they are joined with other agents' actions. Sets of agents that are not effected by each others actions thus take turns to perform their actions a set number of times on initialization. In a grid network of any size a maximum of 6 separate sets of agents whose actions do not effect each other exist, thus at most six of these initialization turns need be taken on initialization. Due to the fact that SARS tuples are saved to file this process really only need take place the first time the particular simulation is ever run. During this initialization phase learning rates are set to an initial high value of 0.9 and utility values are initialized to the value of the immediate reward. When the SARS tuples are being read from file from here on out this initial set of tuples have a learning rate of 0.9 while all other SARS tuples have a learning rate of 0.2. The lower learning rate is to reduce the effects of outlier data as commonly occur in such a stochastic environment.

4.3.2 Get Value

When the MLH Function Approximator getValue method is called the first step is to calculate the hash code for the state. This is done by firstly entering the State Variables into a MLH State Variable Array List. Once the State Variables have been entered the MLH State Variable Array List calls the

hashCode method of each State Variable and then arranges this sequence of hash codes into one single state hash code. The action hash codes are then calculated by calling the Action's hashCode method. Once the State and Action hash codes have been calculated then an ArrayList of Values can easily be retrieved from the values hash map of each Layer. This ArrayList contains one Value for each spatial discount factor. Of the set of Values returned, one per layer, the final Value to be returned can be either the one with the lowest error or the one with the highest granularity, usually being the one from the Layer that is closest to the end of the MLH function approximator's layers array list. In our implementation we use the Value that is associated with the most granular layer as opposed to the one with the lowest error. This is because we set up the MLH so that the most granular layer would be the most accurate layer i.e. our MLH does not have redundant overly granular layers.

4.3.3 Update Values

When the MLH function approximator's updateValue method is called then it in turn calls the updateValue method of each Layer. For each Layer the state and action hash codes are calculated as described in the previous subsection. If no ArrayList object entry exists in the Layer's values array for those hash codes then a new one is created. For each Value in the ArrayList an immediateReward of the corresponding given rewards, a utility of the same value, an updateCount of 1, and an error of Double.MAX_VALUE. If a MLH Value entry does exist then its updateCount is incremented and the error and immediateReward values are updated. The utility values can then be updated according to the Learning Algorithm type being implemented.

4.3.4 Action Selection

The Qoordination Agent's action selection process begins at the conclusion of an update by the Agent calling the Exploration Strategy's getNextAction method. This method determines if a random action should be performed or if the Policy should select the action to be performed. Random actions do not include the action to be chosen if the agent were to follow the policy. Once this decision has been made the Exploration Strategy's ϵ value is incremented by a set amount. Calling the Policy's getNextAction method simply returns the action associated with the highest utility value. To do this the MLH Utility Function is consulted. The MLH Function Approximator retrieves a set of utility values for the state. The action associated with the highest utility value indicates which action should be chosen.

4.4 Conclusion

In this chapter we have described the implementation of three separate components of our research.

The first component of our research whose implementation we have described in this chapter is our simulation based evaluation platform. This evaluation platform is a separate component to Qoordination. This platform is based upon the PTV VISSIM microscopic traffic simulator, which is a popular and dependable method of traffic control evaluation in both academia and industry. With this platform we can evaluate traffic control methods, with a particular focus on evaluating how these control methods are affected by coordination. The platform generates a series of simulated transport networks of increasing size. Generation of these simulated networks is an automated process, as is the process of running the simulations and generating evaluation reports.

The second component of our research whose implementation we have described in this chapter is the generic framework used to develop Qoordination. This framework is not Qoordination but was used to develop the Qoordination agent design and algorithms. The framework was implemented in java. It is flexible because it allows us to experiment with different agent components such as learning algorithm, function approximation method, and exploration strategy. We experimented a lot using this framework and it was invaluable in the development of the final Qoordination design. We experimented a lot with different exploration strategies before we discovered that starting off with a low exploration value and gradually increasing it until an action was performed was actually a much more suitable method of exploration than any other one present in current literature (see section 3.2.9). We experimented a lot with different learning algorithms until we came to the conclusion that Q-Learning would be best for Qoordination as opposed to ADP (see section 3.2.4). We particularly experimented a lot with different function approximators, taking months to finally develop the MLH - using this framework (see section 3.2.6). Much of section 3.2 describes design decisions made when developing Qoordination that were made using the generic framework. Using this framework we could easily modify the design or come up with a new one such as a SARSA agent that uses lookup tables and e-Greedy exploration.

The third component of our research whose implementation we have described in this chapter is the Qoordination agents themselves. The Qoordination agent design was developed using the generic framework but once the design itself was complete they were re-coded in C++ so that they could be directly integrated into the evaluation platform (see section 4.2.1). This is purely a reimplementation of the design that was developed using the framework presented in this chapter.

Chapter 5

Evaluation and Analysis

This chapter presents an in depth evaluation of the Qoordination approach to traffic controller coordination. This evaluation is performed on the simulation-based evaluation platform described in Chapter 4. This chapter begins by presenting the objectives of the evaluation as well as the metrics that have been chosen to measure the performance. A description is then given of the experiments used in the evaluation, which is followed by an in depth analysis of the results obtained from these experiments.

5.1 Objectives

The goal of the evaluation presented in this chapter is to establish how well Qoordination addresses the requirements defined in section 3.1. The ultimate goal of these requirements is to address our research questions that were defined in section 1.3. We have thus set the following objectives for this evaluation:

- **Objective 1:** Assess how well Qoordination agents can coordinate their actions and establish progressive signal systems throughout the main traffic corridors of the transport network. This addresses requirement Req1.
- **Objective 2:** Assess how well Qoordination agents can adapt to dynamic changes in traffic flow levels as well as to the direction of traffic flow. This addresses requirement Req2.
- **Objective 3:** Analyze the learning functionality of the Multi-Layer Hashing (MLH) function approximator to ensure that it functions as it was designed to. This addresses requirement Req3.

- **Objective 4:** Assess how well Qoordination scales i.e. how well it improves network performance as network size increases. This addresses requirement Req7.
- **Objective 5:** Assess Qoordination of intersection controllers that implement various methods of traffic control such as Round Robin, SAT, and a Q-Learning based method.

The reader will note that there are less objectives given here than there are requirements in section 3.1. The reason for this is that a number of the defined requirements were directly addressed through the actual Qoordination design decisions given in Chapter 3. For example, the requirement for robustness to single points of failure is addressed by Qoordination's fully distributed agent based architecture. In unforeseen circumstances such as a number of agents failing due to a power failure the agents that do not lose power can still continue to coordinate their actions and optimize traffic flow thanks to this distributed agent based architecture. Centralized and even a hierarchical architectures do not share this robustness. With these architectures if key intersections lose power then an entire area of intersections or perhaps even the entire network of intersections can no longer coordinate their actions. Thus the full distribution of control provided by the Qoordination architecture enables robustness to single points of failure.

5.2 Metrics

The VISSIM microscopic traffic simulator, on which our evaluation platform is based, provides a number of evaluation variables with which we can analyze the performance of Qoordination. These metrics are as follows:

- **Queue length -** The average approach queue length. Low average queue lengths are indicative of efficiently progressive traffic flows.
- **Waiting time -** The average amount of time that vehicles are waiting in a queue on an approach to an intersection. Low average waiting time is indicative of high traffic control system performance.
- **Number of vehicle stops -** The average number of times a vehicle has to stop as it passes through the simulated traffic network. Low number of vehicle stops is indicative of efficient progressive traffic flows.
- **Speed -** The average speed of vehicles within the simulated traffic network (kph). This is indicative of how well the traffic is flowing through the transport network. High average speed is indicative of high traffic control system performance.
- **Throughput -** The average number of vehicles to pass through an intersection approach per cycle. High throughput is indicative of high traffic control system performance.

- **Traffic volume -** The average number of vehicles within the simulated traffic network at any one time. High traffic volumes are indicative of congestion within the traffic network.

Each of these metrics is averaged over the course of the previous cycle or a specified number of previous cycles as described in section 4.2.2.2. It should be noted that although these metrics cannot measure traffic flow optimization individually they do combine to give a good indication of such.

5.3 Evaluation Transport Networks

The simulated transport networks that are used throughout the evaluation experiments presented in this chapter are automatically generated using our simulation based evaluation platform (see section 4.1.2). Such a generated network consists of a grid of intersections that are spaced equally apart. The intersection layout is uniform throughout the entire network i.e. all intersections have the layout illustrated in Figure 19. In all of the experiments that we present streets are one way streets and all intersections are controlled by intersection controllers i.e. there are no uncontrolled intersections. In many of the scenarios that we present no turns are permitted. These network restrictions mean that there are no opposing traffic flows to cause issues such as those described in section 2.3.1. This also means that the simulation networks, despite having the stochasticity introduced by the VISSIM simulator, are relatively free of noise traffic. The reasons for the topological restrictions outlined here is simply that in order to focus on evaluation of the effects of coordination on traffic control methods the evaluation testbed had to generate networks that varied from each other in size only. Thus it generates sequences of networks that are incrementally enlarged versions of each other (see Figure 43).

The question inevitably arises as to how realistic are these scenarios and would Qoordination's performance within them be a good indication as to whether or not it would actually work in reality.

To begin, the grid network is a common transport network topology in practice e.g. in cities such as Manhattan in the USA. It is also a configuration that is commonly used in evaluating traffic control systems in academia and by traffic engineers (see examples in section 2.3.3). In these networks it is common to feature one way streets and restrictions on vehicle turns. In reality however, even in situations where grid networks with such restrictions are enforced there will inevitably be a certain level of noise traffic. This noise traffic can come from sources such as car parks and side streets. So could Qoordination handle such noise traffic? The answer is that Qoordination would be able to handle such noise traffic as this due to the fact that the Reinforcement Learning (RL) algorithm at the heart of Qoordination i.e. Q-Learning, is a robust algorithm that is capable of handling such noise. There are however limitations on this. In our networks all intersections are controlled by signal controllers. This is because we decided early in the design phase that we would model our environment as being fully observable (see section 3.2.3). As Qoordination agents need to know which intersections are their neighbors the introduction of

uncontrolled neighboring intersection, through which a significant amount of traffic flowed, would hinder a Qoordination agent from learning how to maximize its local reward and thus establish coordination. Partial observability of the environment is out of scope of this thesis but is something that could be addressed in future research. Qoordination can however handle traffic that is permitted to turn, as is shown in experiments 4 and 5 (see sections 5.6.5 and 5.6.6). One thing however that we do not address in any of our experiments is two directional streets, or intersections that must handle opposing traffic flows. In these situations Qoordination would be able to continue to maintain the dynamic progressive signal systems that it forms throughout the network so long as there are no opposing traffic flow directions that are equally dominant. This is because each Qoordination intersection optimizes its offset values based on the direction in which the main flow of traffic runs (see section 3.4.1). If the dominant flow of traffic through an intersection is running from north to south then the intersection agent will establish the progressive signal system in a southbound direction. If multiple non-opposing streams of traffic flow through the intersection then the intersection agent establishes the progressive signal system to cater for this too e.g. if the dominant flows run from north to south and from west to east then the agent establishes the progressive signal system in a south-east bound direction. If however opposing traffic flows are equally dominant then the Qoordination agent struggles to determine the direction in which to establish the progressive signal system, as would you or I in the same situation, e.g. if a clearly dominant flow of traffic runs through an intersection from north to south but the flows of traffic from east to west and from west to east were equally dominant then the agent would constantly change the direction of the progressive signal from south-east to south-west. With regards to the question of the uniform layout of the intersections and whether variations in the intersection layouts would negatively affect Qoordination's performance the answer is that because Qoordination's actions are based solely on the offset between its cycle start times and those of its neighbors the patterns that Qoordination agents learn are not affected by intersection layouts or number of phases. Thus variety in intersection layout should have no effect on Qoordination's learning capabilities.

Taking this information into account we can say that it is reasonable to believe that Qoordination's performance within the networks used for our evaluation are a good indication as to whether or not it would work if implemented in a real transport grid network. Further research would however need to be undertaken in order to take into account uncontrolled intersections within the network i.e. for it to work within a partially observable environment.

5.4 Traffic Control Methods

Throughout the evaluation presented in this chapter Qoordination is applied to intersection agents that implement three different traffic control methods. These methods are Round Robin, a basic SCATS based

control method named SAT, and a single-agent Q-Learning based control method that we developed. Round Robin and SCATS are among the most widely used methods of traffic control globally. The Q-Learning based method provides for a good example of a learning traffic control method.

5.4.1 Round Robin

The Round Robin method of traffic control is an uncoordinated version of pre-timed control that is described in section 2.3.2.1. This method assigns a set duration for each phase and circulates through these phases in a set sequence. This is not an adaptive approach to traffic control so these phase durations and phase sequences are not altered throughout the course of the simulations. As mentioned in section 4.1.2.1 the intersections that are implemented in the evaluation platform have the same layout as the one illustrated in Figure 19. Each intersection controller runs a six phase cycle, illustrated in the following ring diagram:

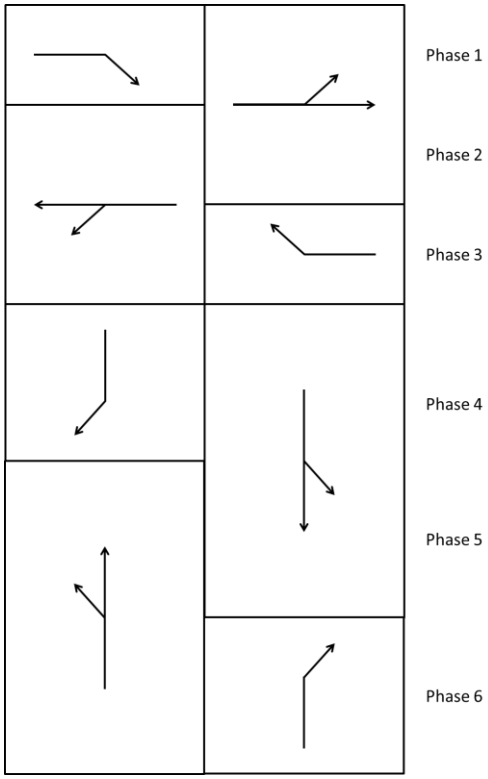

Figure 46 Six phase ring diagram

In Phase 1 all traffic flowing from the west approach receives a green light while traffic approaching from all other directions is stopped. In Phase 2 the dedicated right turn flowing from west to south is stopped and traffic from east to west and from east to south is given a green light. In this phase traffic

flowing from both east and west are given green lights with the exception of dedicated right turns. In Phase 3 all traffic flowing from the east approach receives a green light while traffic approaching from all other directions is stopped. Phases 4 to 6 apply this same pattern to the traffic flowing from the north and the south. For phases two and five i.e. those that don't have any dedicated right turns, we assigned the default phase length i.e. 60 seconds. For all other phases we assigned half this duration i.e. 30 seconds each. The end of each phase consists of a change interval of 3 seconds and an all red interval of one second. Each cycle thus has a total duration of 240 seconds. We have found that these phase durations and sequence are enough to keep the intersection relatively clear of traffic under medium traffic flow levels.

5.4.2 SAT

SAT (Richter, 2006) is a basic adaptive traffic control method that is based upon the SCATS traffic control system. This approach modifies phase durations in steps (steps of 6 seconds in our implementation) based on traffic saturation levels at the intersection. Saturation is the effective use of the total available green time. SAT aims at maintaining saturation levels *SL* as close to 90% as possible. The target maximum throughput *TMT* for phase *p* is defined as follows:

$$TMT_p = \frac{MT_p}{SL} \tag{40}$$

where:

- MT_p is the maximum throughput of the approaches that receive a green light during phase *p* over the previous cycle

Phase *p*'s target length TL_p is then set by multiplying TMT_p by a scaling factor. If TL_p is greater than *p*'s current length then *p*'s length is increased by a specified amount, which we have set to 6 seconds. If TL_p is lower than *p*'s current length then *p*'s length is decreased by this same amount. A phase length must not exceed a set maximum value, which we have set to 90 seconds. It should also not go under a set minimum value, which we have set to 6 seconds. SAT's maximum cycle length is calculated using the following formula:

$$Min_{pt} \times Max_{cf} \times p \tag{41}$$

where:

- The minimum phase time Min_{pt} is set to 6 seconds
- The maximum cycle factor Max_{cf} is set to 1.5
- The number of phases p is 6

After the modifications to the phase lengths have been made the total cycle length must not exceed this maximum cycle length. If this has occurred then SAT cycles through the phases and reduces them by 1 second each until the cycle is below the maximum allowable length.

5.4.3 Q-Learning Based Traffic Control

Our single-agent Q-Learning based control was also developed using our traffic control framework described in 4.2. Like Qoordination this Q-Learning based traffic control method uses a MLH utility function. As state variables this method uses phase durations and vehicle throughputs. Vehicle throughput is measured per phase, where a phase's throughput is the highest throughput measured on the approaches assigned to that phase over the course of the past cycle. A total of 12 state variables are thus used. The MLH contains 15 layers and coarseness values for both phase length and throughput are set between 6 and 90. This assumes that no more than an average of one vehicle per second can pass through any approach during any phase. Actions consist of phase duration modifications. Thus the agent can increase or decrease each phase length or keep the phase lengths the same. Phases can be increased or decreased in steps of 6 seconds up to a maximum of 90 seconds or to a minimum of 6 seconds. One action is performed per phase at every time step. This means that each phase has the opportunity to have its duration changed at each time step. A separate reward is calculated per phase during each update. These rewards are calculated by getting the distance between the phase's duration and a calculated phase length goal that aims at having saturation levels at 90%. This calculated phase length is similar to the one calculated by SAT. Phase skipping is not allowed and an ϵ-Greedy exploration strategy is implemented. Updates occur every two cycles although state variable values and rewards are calculated based only on one previous cycle.

5.5 Parameters

This section specifies the values of a number of the more important Qoordination parameters as they are set during this evaluation unless stated otherwise.

- Learning rate $\propto$: 0.2
- Temporal discount factor γ: 0.8
- Initial ϵ value: 0.0
- Maximum ϵ value: 0.3
- ϵ increase rate: 0.01
- Number of layers: 45
- Minimum offset coarseness value: 12

- Maximum offset coarseness value: 540
- Offset modification step size: 12

A low learning rate such as 0.1 or 0.2 is quite typical of an agent that operates within a dynamic or multi-agent environment (Sutton & Barto, 1998) as this enables efficient handling of outlier data and the stochasticity of traffic data. The temporal discount factor of 0.8 leads to relatively far sighted agents. The reasoning behind setting the ϵ related values as they are set is given in section 3.2.9. The maximum offset coarseness value is set to 450 because this is the maximum cycle length i.e. 90 * 6. All of these values were decided upon after experimentation with a range of different values. These are the values that we found were most suitable.

5.6 Experiments

This section gives a description of the different scenarios that are implemented in evaluating Qoordination's performance. Each scenario aims to accomplish one or more of the evaluation objectives outlined at the beginning of this chapter.

5.6.1 Experiment 1 – Increasing Network Size and Static Traffic Flow Levels

The purpose of this experiment is to show that Qoordination can establish progressive signal systems in networks of static traffic flow direction and static traffic flow levels. In this experiment we vary the traffic flow levels from low to medium to heavy. Throughout the duration of each simulation however the traffic flow level are kept constant. This experiment also shows the effects of Qoordination on networks of different size. This experiment is conducted within a series of increasingly larger transport networks. The smallest of these networks is a 1x1 network i.e. a single intersection. The largest of these networks is a 5x5 network i.e. a network of 25 intersections. 2x2, 3x3, and 4x4 networks are also used in the experiment. The purpose of this is to meet objective 4 i.e. analysis of how well Qoordination scales with increasing network size. It also meets objective 5 i.e. assess the effects of Qoordination of various methods of traffic control. In this experiment traffic flows from the west of the network to the east and from the south of the network to the north. All roads in use are thus one way roads. No left or right turns are permitted. Light traffic flow levels permit 360 vehicles per hour into the network by each of the north bound and east bound roads. Medium traffic flow levels permit 650 vehicles per hour and heavy traffic flow levels permit 950 vehicles per hour through each of these same roads. Each simulation has a duration of 21,600 seconds, i.e. 6 hours. Also included in this experiment are the results of an actuated control method. These results are included for the single intersection and non-coordinated aspect of this

experiment but not in the coordinated aspect of this experiment because, as will be explained in section 5.6.1.4 we discovered that actuated control is not suitable for coordination.

5.6.1.1 Comparison of Traffic Control Methods in a Single Intersection

Let us begin by comparing the performances of the different traffic control methods on a single intersection. In this situation there is not only no possibility of coordination but also no possible benefit of it. As stated previously the different methods of traffic control used are Round Robin, SAT, and a Q-Learning based control method.

5.6.1.1.1 Results and Analysis

In this sub-section we present the results of this experiment and analyze these results. We further analyze these results and discuss observed patterns and findings in sub-section 5.6.1.1.2.

5.6.1.1.1.1 Medium Traffic Flow Levels

The following diagrams illustrate this comparison under medium traffic flow levels:

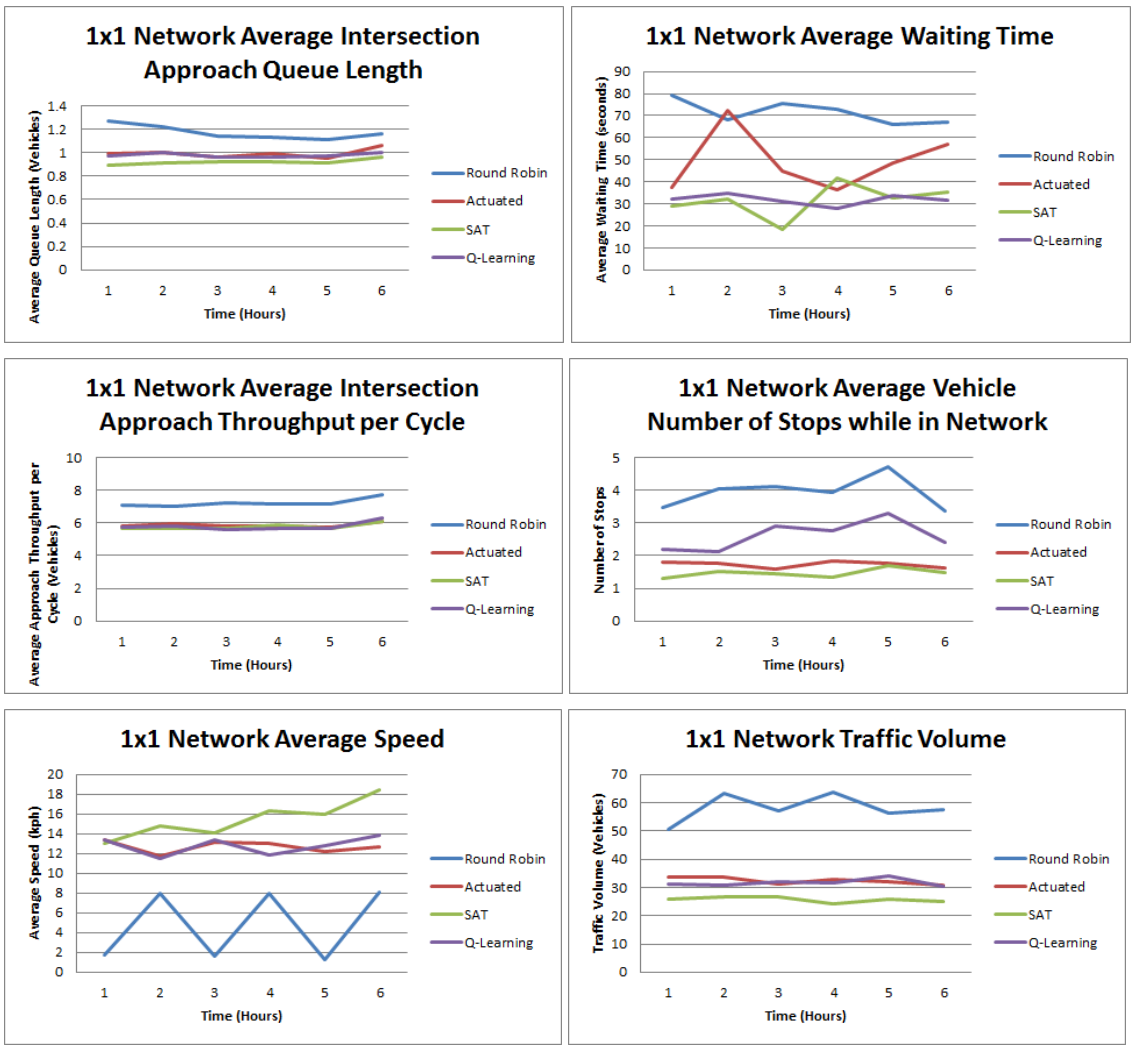

Figure 47 Traffic control method performance comparison, single intersection, medium traffic flow levels

Along the x axis of each of these diagrams we can see the number of hours that have expired since the beginning of the simulation. Along the y axis of each of these diagrams we can see a measurement of the specific metrics represented in the individual diagrams e.g. average queue length, average waiting time, and average vehicle throughput. Our first observation of these diagrams is that on average Round Robin performs worst with regards to the different evaluation metrics while SAT performs best of the methods evaluated. Actuated and Q-Learning based control tend to perform similarly and somewhere in-between Round Robin and SAT.

We can see from the first of the diagrams in Figure 47 that under medium traffic flow levels SAT achieves slightly lower queue lengths than actuated and Q-Learning based control. Round Robin queue lengths however stand out as being higher than these. In this diagram we can see that the general trend is that the average queue lengths stay roughly stable. For actuated, SAT, and Q-Learning based control

there is very little variation in average queue lengths over the course of the experiment. For Round Robin there is a slight decrease of approximately 0.1 vehicles over the course of the first hour, which is not a significant decrease, from which point on there is very little fluctuation. One reason for such low levels of fluctuation in such a stochastic environment is that the queue lengths are being averaged over the course of the previous cycle length. We theorize that the main reason however for such a lack of fluctuation in average queue length is that, as has been mentioned before e.g. in section 3.2.3, because the induction loop sensors are spaced 25 meters apart the agent can only detect approximately five average length vehicles at a standstill on any one of the intersection's approaches. Under medium traffic conditions the queue lengths on the approaches that are receiving traffic are longer than or equal to five vehicles a lot of the time. This would lead to average queue lengths of between four and five vehicles if traffic was flowing in all directions. In this experiment traffic is only flowing from south to north and from west to east, thus reducing the average queue lengths for the intersection approaches.

In the second diagram in Figure 47 we can observe that different traffic control methods do perform significantly differently from each other with regards to average vehicle waiting time. Both the metric of average queue lengths the metric of average vehicle waiting time are based upon readings that are limited to roughly 5 average length vehicles. The metric of average waiting time however is less limited in representing the actual state of the traffic flow because it also takes into account how long these vehicles have been waiting in the queue. This does not make it more suitable for the metric on which we base Qoordination's ability to coordinate, for reasons described in section 3.2.12, but it does make it a more suitable for evaluation purposes. In this diagram we can see that Round Robin average vehicle waiting times are consistently over twice as high as both SAT and Q-Learning based control. Actuated control waiting times are a bit more irregular though they tend to be closer to those of SAT and Q-Learning based control than to Round Robin. This is a clear indication that Round Robin performs worst of the tested traffic control methods for a single intersection under medium traffic flow levels. SAT and Q-Learning based control perform best. Actuated control on the other hand acts a little more irregularly. These results confirm what we would have expected to see from this experiment based upon the designs of the different traffic control methods.

In the third diagram of Figure 47 we can see the average throughput levels achieved by the four tested traffic control methods. We can observe that these readings stay roughly level for each of the different traffic control methods throughout the duration of the experiment. We put this down to the fact that there was no major change in traffic flow levels throughout this experiment and that minor fluctuations in traffic flow levels that were introduced by VISSIM were smoothed out by averaging the results over the course of the previous cycle. In this diagram we can observe that actuated, SAT, and Q-Learning based control methods all achieve similar results with regard to this metric. Round Robin however achieves roughly 15% higher than this. This particular result is somewhat surprising as we would have expected Round Robin to have a lower throughput than the others, particularly since SAT and the Q-Learning

based method aim to maximize throughput. We speculate that the build-up of Round Robin queue lengths leads to a higher throughput level per cycle than if the queue lengths were kept down, as is done with the other methods. This highlights to us that the method that we use to calculate throughput does not accurately portray the true state of traffic flow within the transport network. A more accurate portrayal of the true state of traffic flow could possibly however be achieved if we were to have measured the average approach throughput over a fixed time span. This fixed time span should be the same for all traffic control methods and should not change as the cycle length changes. It should still be long enough to smooth the data. Perhaps a duration of 60 or 120 seconds would suffice.

The fourth diagram of Figure 47 illustrates the performance of the different traffic control methods with regards to the average vehicle number of stops metric. This metric is calculated using data outputted by VISSIM itself as part of its .fzp files, as opposed to being calculated by the outputs of our Qoordination agents. Because of this there is slightly more fluctuation of the metric for the control methods over the duration of the experiment. We can observe from these results that SAT achieves the lowest average number of vehicle stops, followed closely by actuated control. The Q-Learning based method performs quite worse in this category, with an average number of stops twice that of SAT for a significant percentage of the simulations. Round Robin comes off worst in this category with approximately 30% higher average number of vehicle stops than the Q-Learning based method.

The fifth diagram of Figure 47 which illustrates average vehicle speed shows quite an interesting pattern. It shows fluctuations in the results of the various traffic control methods over the course of the experiment. This is particularly highlighted with Round Robin as it results in a significantly fluctuating zig-zag pattern. One thing to note about the average speed metric is that, similar to the number of stops metric and the traffic volume metric, it is calculated based on VISSIM's .fzp output files. This means that it does not experience the smoothing effect that occurs due to the limitation of a maximum queue length of approximately 5 average length vehicles experienced by the first three metrics. Additionally, vehicle speed by nature is a very dynamic variable. We also do not average this metric over the duration of the previous cycle or specified number of cycles. It is for these reasons that this metric seems to fluctuate so much. With regards to this metric Round Robin again achieves significantly lower results than the other methods. Actuated control and the Q-Learning based method achieve similar results with SAT achieving significantly higher than even these. Although actuated, Round Robin, and Q-Learning based control do not show any long term trend of increasing or decreasing performance with regards to this metric we do see that SAT results seem to steadily increase over time. This is in part due to random fluctuations caused by the stochasticity of the environment. It is also due to the fact that over time SAT adjusts its phase lengths to the optimum settings, resulting in higher average vehicle speeds over time.

In the final diagram of Figure 47 we can observe the performance of the four tested traffic control methods with regards to the traffic volume metric. This metric is also calculated using data taken from VISSIM's .fzp output files. Traffic volume however is a much less dynamic variable than average vehicle

speed and so we do not see as much fluctuations as we did in the previous diagram. In this diagram we can see that the Round Robin method leads to significantly higher traffic volume levels than the other three methods. These resulting traffic volumes are in fact up to twice as high as for the other methods. Whereas actuated control and the Q-Learning based method maintain similar traffic volume levels SAT again performs visibly better than the rest.

From this initial analysis we can see that in most categories actuated control and the Q-Learning based method perform similarly in medium traffic flow levels. Whereas SAT performs consistently equal to or better than them the Round Robin method performs significantly worse. The exception to this is the category of vehicle throughput, which we have resolved is not calculated in such a way as to accurately portray the true state of traffic flow within the system.

5.6.1.1.1.2 Low Traffic Flow Levels

We will now make this same comparison of traffic control method performance during low traffic flow levels.

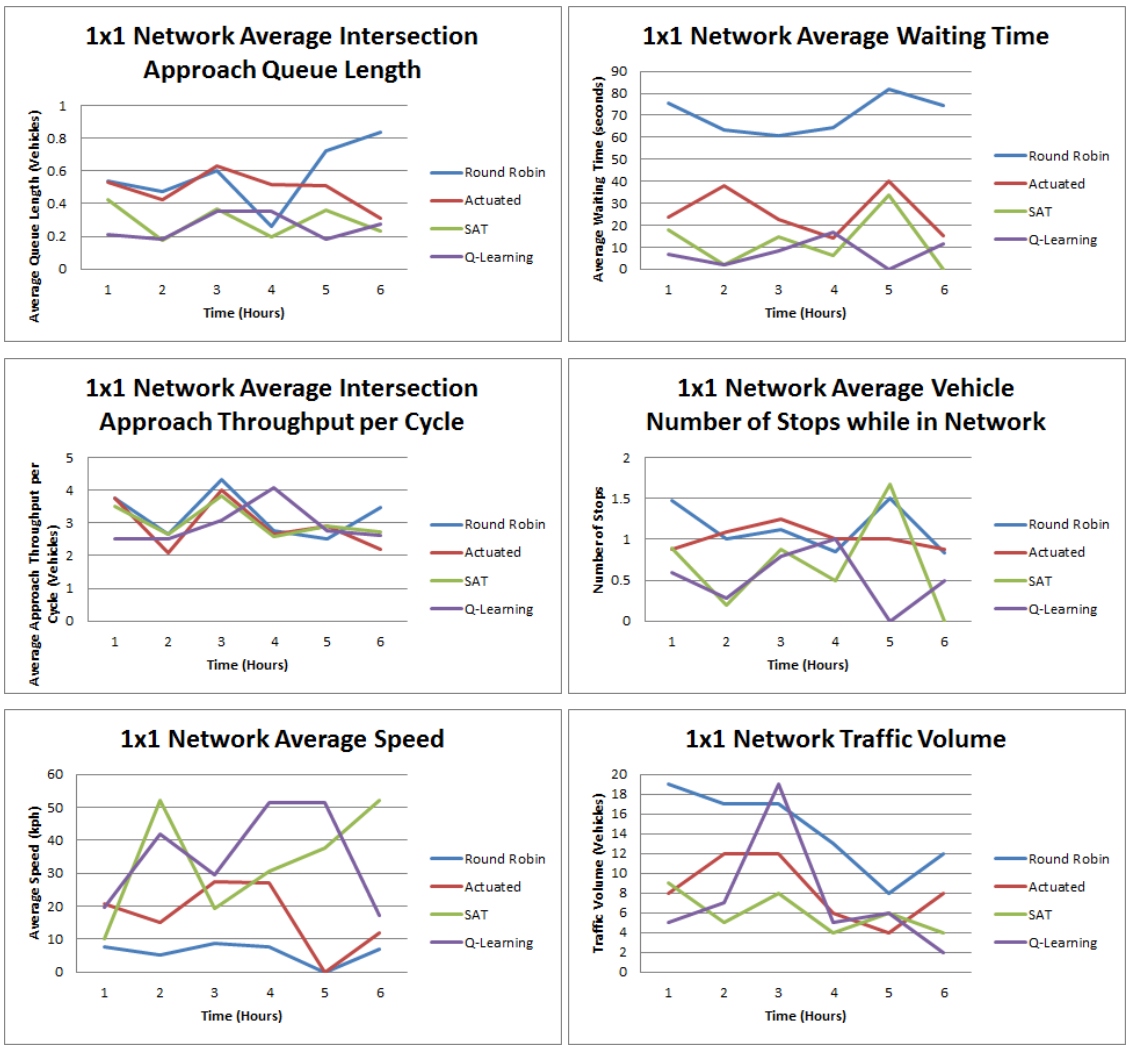

Figure 48 Traffic control method performance comparison, single intersection, low traffic flow levels

Under low traffic flow levels the differences in traffic control method performance in most categories is not as clearly visible as under medium traffic flow levels. This is due to the fact that many of the results fluctuate significantly throughout the course of the experiment, much more so under low traffic flow levels than under medium traffic flow levels. We theorize that the reason for this increase in fluctuation is because the low traffic flow levels lead to queue lengths that are typically less than five vehicles. As we mentioned in the previous sub-section the queue length limitation of roughly 5 average length vehicles due to the 25 meter apart spacing of approaches' induction loops leads to a more steady average of a full queue. With low traffic flow levels the measurable queue lengths fluctuate much more as they are not usually full.

In the first diagram of Figure 48 we can observe these greater fluctuations than under medium traffic flow levels. In this diagram we can see that both SAT and the Q-Learning based method have similar

average queue lengths. Round Robin and actuated control have higher average queue lengths than these. All of these average queue lengths are much lower than what we had observed under medium traffic flow levels.

In the second diagram of Figure 48 we see performance of the four tested traffic control methods with regards to the metric of average vehicle waiting times. In this category SAT and Q-Learning based control again perform quite similarly. Their similarity is slightly masked by the additional fluctuations caused by the lower traffic flow levels. Actuated control maintains waiting times slightly higher than SAT and Q-Learning based control but the Round Robin method performs significantly worse, with average waiting times of almost three times that of actuated control.

The third diagram of Figure 48 shows that all four tested methods of traffic control have similar throughputs under low traffic flow levels. In this diagram we can again observe the fluctuating results that are caused by the low traffic flow levels.

With regards to the metric of average number of vehicle stops the four tested methods of traffic control again perform quite similarly, with SAT and the Q-Learning based method tending to have similar and lower number of stops than Round Robin and actuated control early on in the experiment.

The fifth diagram of Figure 48 illustrates the tested traffic control methods' performance with regards to the metric of average vehicle speed. Round Robin performs worst in this metric. It is followed by actuated control and then the Q-Learning based method. SAT performs better than Round Robin and actuated control, yet due to fluctuations which we ascribe to the low traffic flow levels it varies between performing better at times and worse at times than Q-Learning based control.

Round Robin also performs significantly worse than the other methods with regard to network traffic volume, as illustrated in the final diagram of Figure 48. The other three tested methods perform similarly with regards to this metric, as can be seen in this diagram despite the fluctuations caused by the low traffic flow levels. Note that these fluctuations are also more noticeable in this diagram then in the one from the medium traffic flow level scenario because of the scale difference. Even the highest of the traffic flow volumes in the low traffic flow level scenario (Figure 47) are lower than the lowest of the traffic flow volumes in the high traffic flow level scenario (Figure 48).

From the results provided in this sub-section we can see that SAT and Q-Learning based control tend to perform best under low traffic flow levels, with Round Robin typically performing worst.

5.6.1.1.1.3 High Traffic Flow Levels

Let us now consider the case of high traffic flow levels.

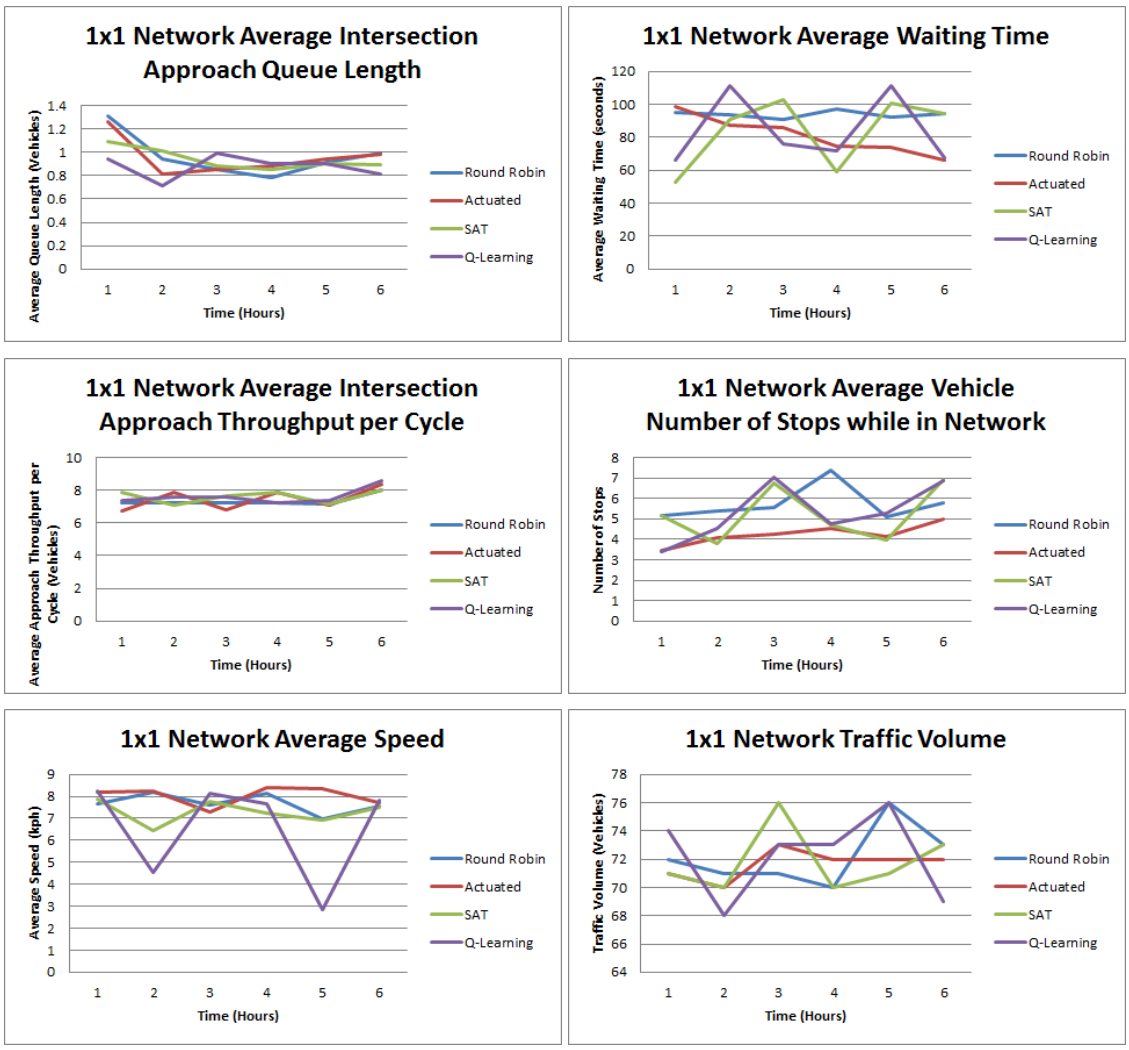

Figure 49 Traffic control method performance comparison, single intersection, high traffic flow levels

If it was true that under low traffic flow levels the differences between the performances of the traffic control methods is not as clearly visible then this is even more true in the case of high traffic flow levels. The reasons for this however are different under high traffic flow levels than under low traffic flow levels. Under low traffic flow levels it was difficult to make the comparison because of additional fluctuations in the readings. Under high traffic flow levels it is difficult to make the comparison because the intersection is constantly at maximum capacity despite the different traffic control methods being used.

With regard to average queue lengths (first diagram of Figure 49) and vehicle throughput (third diagram of Figure 49) all methods perform similarly. As mentioned, this is because there is constantly a full queue on each approach containing any traffic regardless of the traffic control method in use.

We can however see that with regards to vehicle waiting time (second diagram of Figure 49) actuated control performs consistently better than Round Robin, with the other two methods acting somewhat more dynamically.

In the fourth diagram of Figure 49 we can see the same pattern of actuated control performing consistently better than Round Robin with the other two traffic control methods performing more dynamically. We theorize that the results of SAT and Q-Learning based control fluctuate more than the other methods in this scenario because they try to adapt their timing plans to improve traffic flow. This helps for a little while but ultimately they have to increase their phase lengths to the maximum allowable values again because the traffic flow levels are consistently so high and congestion is inevitable. Round Robin cannot adjust its timing plans and is thus consistently congested. Actuated control consistently extends its phase lengths to the maximum limit as there is never any gap in the traffic flow which would allow it to shorten them.

With regards to the final two metrics (the final two diagrams of Figure 49) all traffic control methods perform quite similarly due to the fact that they are all at full capacity.

From the results described in this sub-section we can see that all traffic control methods perform quite similarly due to the fact that they are all at maximum capacity. That being said, SAT and Q-Learning based control tend to have more fluctuating performances than the other two methods. With regards to the metrics of average waiting time and average number of vehicle stops actuated control has proven to perform consistently better than Round Robin.

5.6.1.1.2 Concluding Analysis

In this experiment scenario we have shown how each of the four traffic control methods under examination, namely Round Robin, actuated control, Q-Learning based control, and SAT, perform on a single intersection. Their performance is with regards to a number of metrics which have been described in section 5.2. We have also shown how these methods perform with regard to these metrics under low, medium, and high traffic flow levels. Qoordination has not been examined in this scenario, nor has any other form of coordination, as the scenario deals with a single intersection, whose agent has no other intersection agents to coordinate with. We have found that the performances of these four traffic control methods are easiest to compare to each other under medium traffic flow levels. In low traffic flow levels the results tend to fluctuate due to the fact that the queue lengths vary significantly between empty and full at any one time. Queue lengths in medium traffic flow levels on the other hand tend toward being full a lot of the time, which has a smoothing effect on the different metrics, as described in section 5.6.1.1.1.2. These fluctuations could perhaps be smoothed by running the experiments a number of times more than done in our experiments and then averaging the results. In high traffic flow levels the results tend to be quite similar across the different traffic control methods due to the fact that the traffic queues are constantly at maximum capacity. Despite these challenges we have been able to observe that in general

Round Robin performs worse than the three other traffic control methods taken into consideration. This applies across all metrics except one, namely throughput per cycle. As discussed in section 5.6.1.1.1.1 the way that throughput is calculated does not accurately portray the true state of traffic flow within the transport network. In our evaluation scenario that investigates the different traffic control methods under medium traffic flow levels we have been able to observe that SAT generally performs better than the other methods. Under low and medium traffic flow levels however it tends to perform comparably to both actuated and Q-Learning based control.

As we have observed thus far, the throughput metric does not accurately portray the true state of traffic flow within the transport network. For this reason we will not consider it any further in the remainder of our experiments.

5.6.1.2 Effects of Network Size on Uncoordinated Traffic Control Methods

Having looked at the performances of the various methods of traffic control in a single intersection let us now consider how these methods are affected by increasing network size when no form of coordination is applied. Thus Qoordination is not being evaluated in this scenario. We are comparing the performance difference between the different traffic control methods without any coordination considered. Each intersection controller operates independently of any of the other controllers in the networks and is even unaware of their very presence within the environment.

5.6.1.2.1 Results and Analysis

In this sub-section we present the results of this experiment scenario and analyze these results. We further analyze these results and discuss observed patterns and findings in sub-section 5.6.1.2.2.

5.6.1.2.1.1 Medium Traffic Flow Levels

Let us first consider increasing network size of networks under medium traffic flow levels.

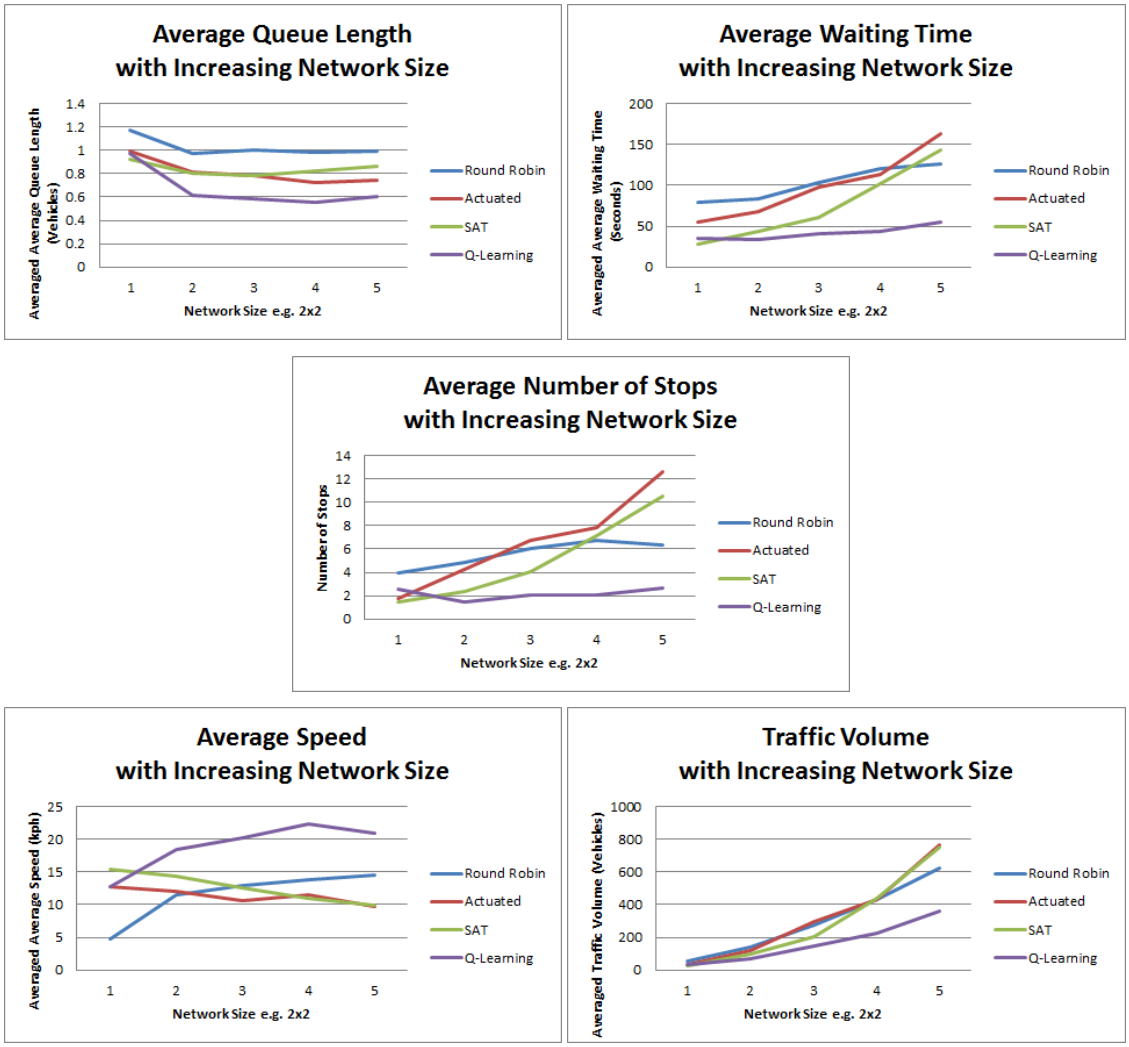

Figure 50 Performance comparison with network size increase, no coordination, medium traffic flow

The diagrams presented in the remainder of our evaluation are significantly different from those presented in the previous evaluation scenario of section 5.6.1.1. In the previous diagrams the x axis represented the passing of time since the beginning of the simulation. In the diagrams in the remainder of the evaluation experiments however the x axis represents the network size. In these experiments the value on the x axis of 1 represents the averaged results of the associated diagrams from the single intersection scenario in section 5.6.1.1. A value of 2 on the x axis represents a network with 2x2 connected intersections such as the one shown in Figure 63. A value of 3 represents a 3x3 network, such as shown in Figure 65and so on. As with the previous diagrams, along the y axis of each of these diagrams can be seen a measurement of the specific metrics represented in the individual diagrams e.g. average queue length and average waiting time.

From the first diagram of Figure 50 we can see that for all traffic control methods tested the average queue lengths are higher on a single intersection then they are in larger networks. For any of the traffic control methods the average queue lengths then remain roughly stable as the network size increases from 2x2 intersections upwards. The reason that the average queue lengths for each of the control methods is higher in a single intersection scenario is that all the traffic that arrives on the approaches of the single intersection arrive randomly thanks to VISSIM. In larger networks traffic arrives on the approaches at the border of the network randomly but is then arranged into platoons of vehicles by passing through the first intersection that they encounter in the network. Intersections that are not at the edges of the simulated transport network only ever encounter platoons of vehicles. Seeing as there is no coordination between intersection agents the offsets between them stay relatively static. Initially this offset is 0 seconds. An offset of 0 seconds between intersections that are placed roughly 100 meters apart leads to much of the traffic flowing through one intersection to be also met with a green light in the subsequent intersection. Thus in this scenario platooned traffic leads to significantly lower average queues than randomly arriving vehicles. In this diagram the queue lengths of all intersections within their perspective network size categories are averaged. This diagram thus highlights that the introduction of platoons of vehicles into the transport network reduces the average queue length.

We can observe from this first diagram that Round Robin consistently reaches the highest levels of average queue lengths when compared to the other methods. This was expected. What is somewhat less expected is that Q-Learning based control consistently achieves lower queue lengths than both actuated control and SAT. We would have expected SAT to perform best in this metric as it had done in the single intersection scenario. As we can see, actuated control and SAT performed quite similarly, with SAT performing slightly better than actuated control as the network size increases. Both actuated control and SAT are quite dynamic methods of control, with phase lengths changing quicker than Q-Learning control. We theorize that Q-Learning based control performs better than the three other methods when it is not modifying its offset because the average cycle length of the Q-Learning based signal controllers remain more similar to each other, thus keeping them somewhat coordinated, even if that level of coordination does only allow for simultaneous progression. With actuated control and SAT changing phase lengths and thus cycle lengths at a more rapid pace their signal controllers soon become completely uncoordinated with their neighbors. With Q-Learning based control performing better than the other control methods in this metric it will have a knock on effect of it performing better than them in all other metrics.

In the second diagram of Figure 50 we can see that as the network size increases so too does the average vehicle waiting time for each of the four traffic control methods. A particularly interesting phenomenon that we can observe in this diagram is that although Round Robin has higher average waiting times than actuated control and SAT for single intersections the latter methods' average waiting times increase at such a rate as network size increases that within the 5x5 intersection network Round Robin has lower waiting times than the both of them. The Q-Learning based method however maintains a

similar rate of increase as Round Robin. This confirms our theory that Q-Learning based controllers remain somewhat coordinated with each other because their average cycle lengths do not vary much from each other.

In the third and fourth diagrams of Figure 50 we can see that regarding the metrics of average number of vehicle stops and average speed Round Robin performs worse in a single intersection than actuated control and SAT but then out passes them as the network size increases. Again we speculate that this pattern is caused by the fact that Round Robin intersections have the same cycle length and thus remain in a state of simultaneous progression in this scenario. The other approaches do not remain in such a state of coordination and times between the starts of their cycles vary greatly, resulting in worse performance. Due to the fact that the Q-Learning based method performance also decreases at a rate similar to Round Robin's this would insinuate that its intersections also maintain similar cycle lengths, again confirming our theory.

In the final diagram of Figure 50 we can see that Q-Learning based control manages to keep traffic volumes down better than the other approaches with increasing network size. This is followed by Round Robin and then actuated control and SAT. Round Robin again however increases its performance with regards to this metric as network size increases and in comparison to actuated control and SAT.

5.6.1.2.1.2 Low Traffic Flow Levels

Let us now consider the same comparison under low traffic flow levels.

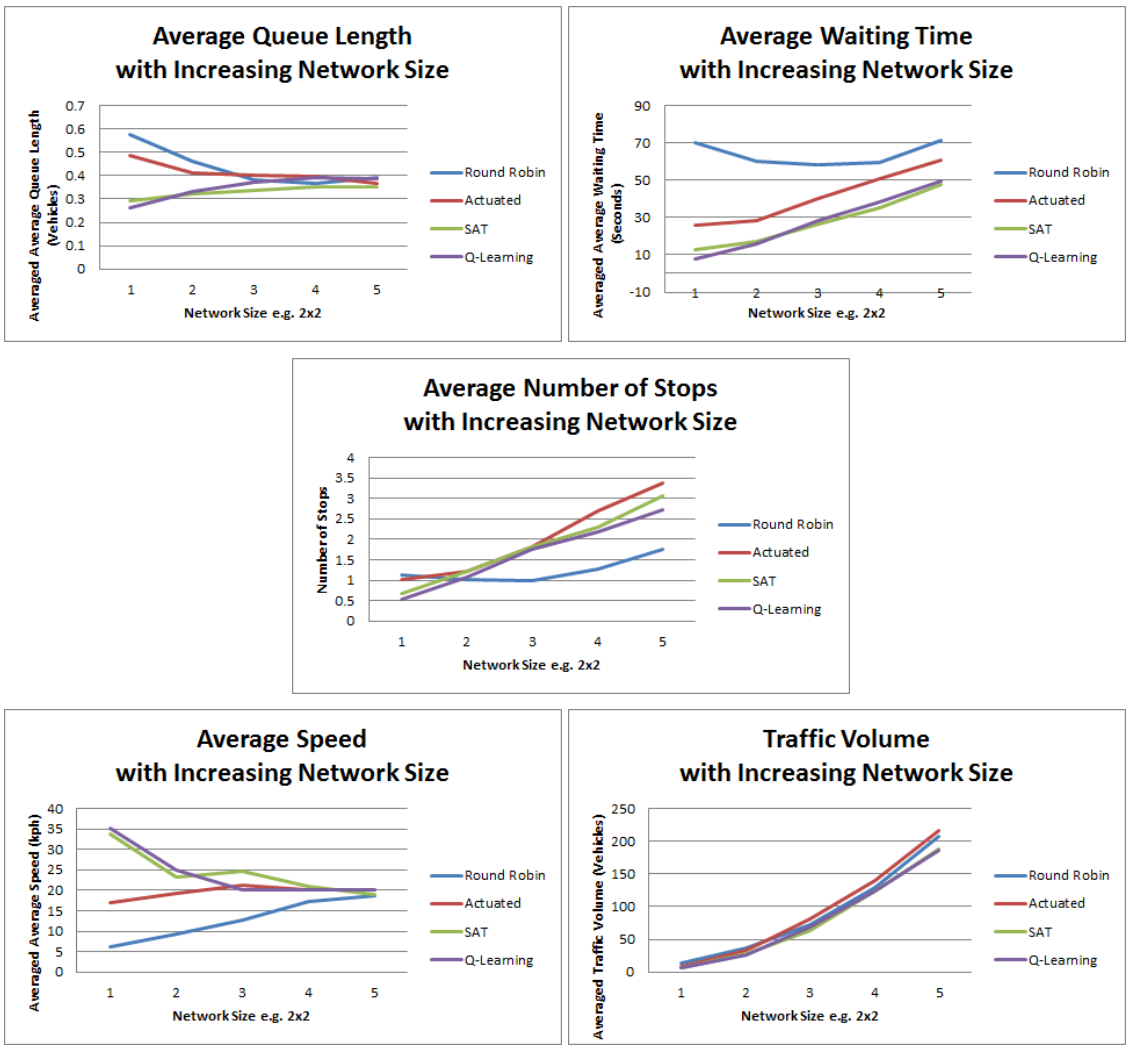

Figure 51 Performance comparison with network size increase, no coordination, low traffic flow

Our first observation from the above diagrams is that as the network size increases the performances of each of the uncoordinated traffic control methods tends to converge and then stay steady for the average queue length metric i.e. the first diagram of Figure 51. This contrasts the same metric under medium traffic flow where the control method's performances went through an initial drop and then stayed stable but showed no sign of convergence. The reasons for such a convergence are unclear. It does however seem to have an effect on the methods' performances with regards to some of the other metrics i.e. average waiting time and average vehicle speed.

The second diagram of Figure 51 shows that average vehicle waiting times increase steadily with increasing network size for actuated control, SAT, and the Q-Learning based control. This does not however seem to happen with Round Robin. Round Robin waiting times seem to initially decrease but then seem to turn and increase as network size increases beyond 4x4 intersections. This pattern is

somewhat similar to the convergence pattern displayed in the first diagram if we take into consideration the first two medium traffic control diagrams of Figure 50. With regards to Figure 50, in the first diagram all performances remain constant while in the second all average waiting times increase as network size increases. Regarding Figure 51 in the first diagram the performances converge and then stay roughly constant while in the second all average waiting times converge to some degree and then increase as network size increases. The initial decrease in Round Robin average waiting time might thus be an effect of the convergence pattern. Throughout the second diagram the average waiting times of each traffic control method do however roughly maintain their ranking i.e. they do not completely converge. SAT and the Q-Learning based method consistently achieve the lowest waiting times. Round Robin consistently achieves the highest waiting times while actuated control achieves waiting times half way between those of Round Robin and the other two methods.

The number of stops increases at a fairly steady rate as network size increases for actuated control, SAT, and Q-Learning based control as can be seen in the third diagram of Figure 51. Similar to what happened with waiting time Round Robins number of stops seems to decrease slightly at first before it starts to rise.

The fourth diagram of Figure 51shows again a convergence of the traffic control methods' performances. The reasons behind this we can assume are the same as have been discussed regarding the queue length metric.

As can be observed in the final diagram of Figure 51 the traffic volume levels increase fairly uniformly across all traffic control methods as network sizes increase. This is as expected because as the network sizes increase so do their capacity to contain vehicles and thus a large network will obviously contain on average many more vehicles at any one time than a small network.

5.6.1.2.1.3 High Traffic Flow Levels

We will now look at the effects of increasing network size in heavy traffic flow levels with no coordination.

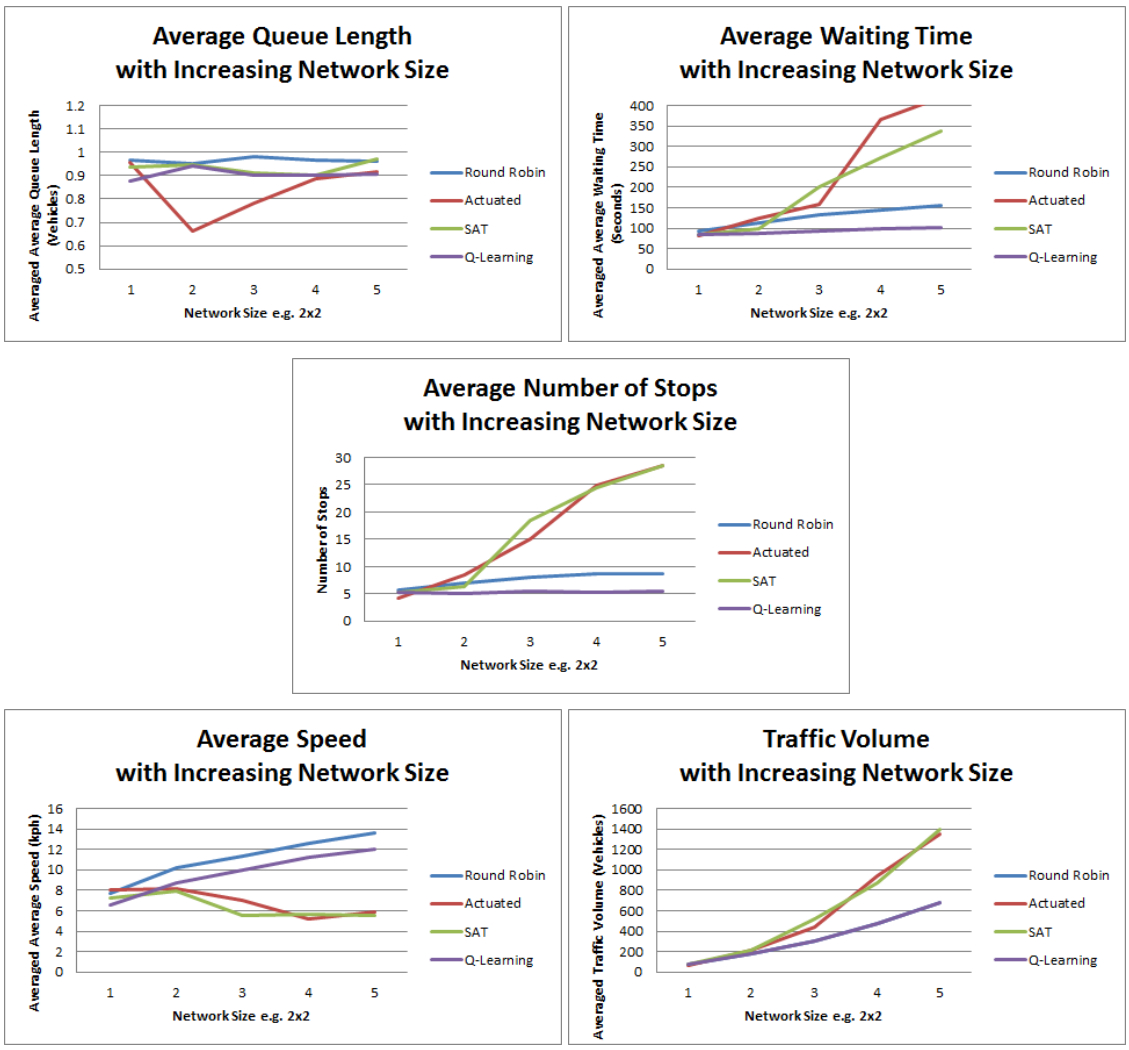

Figure 52 Performance comparison with network size increase, no coordination, high traffic flow

Our first impressions from these diagrams is that under higher traffic flow levels the difference between the more static control methods and the more dynamic control methods is accentuated. We have heretofore theorized that both Round Robin and Q-Learning based control remain somewhat coordinated due to their lack of / slow change in offsets between their cycle start times. This coordination is simply due to the fact the controller cycle lengths remain somewhat the same lengths and it only likely enables simultaneous progression. SAT and actuated controllers do not remain coordinated throughout this scenario due to their dynamic natures. We would thus be lead to believe from these diagrams that coordination is of particular benefit to the performance of the traffic control methods under higher traffic flow levels.

In this scenario the average queue lengths stay somewhat steady with increasing network size for Round Robin, SAT, and the Q-Learning based method. This can be seen in the first diagram of Figure

52. This steadiness is likely due to the fact that under high traffic flow levels the queues are at maximum capacity most of the time. Queue lengths for actuated control however seem to drop initially and then turn and increase after networks of more than 2x2 intersections. We surmise that this occurs because the heavy traffic flow lead all actuated controllers to assign maximum green time to each of the approaches with traffic on them immediately after the simulation was started. This kept the controller cycle lengths equal. While the cycle lengths were kept roughly equal a level of coordination may have been established among the controllers, which combined with the long phase lengths led to lower average queue lengths. As the network size increased this unintentional implicit coordination could not be maintained and thus broke down.

In the remainder of the diagrams in Figure 52 the same pattern emerges again and again. Q-Learning based control performs best in most metrics with Round Robin not far behind. Round Robin however performs best with regards to the metric of average vehicle speed with Q-Learning based control not far behind. SAT and actuated control perform similarly with regards to all metrics, both of which however perform significantly worse than Round Robin and Q-Learning based control. We would speculate that SAT and actuated control perform so similarly in all metrics because the heavy traffic causes them to keep the phase lengths of approaches with traffic on them at a maximum at all times, with all other phase lengths being set to a minimum. Under heavy traffic flow levels the intersections are thus constantly congested with the apparent solution being coordination.

5.6.1.2.2 Concluding Analysis

In this scenario we have shown how each of the four traffic control methods under examination, namely Round Robin, actuated control, Q-Learning based control, and SAT, perform in networks of various size. Network size ranges from a single intersection up to a network of 5x5 connected intersections. The control methods' performance is with regards to a number of metrics which have been described in section 5.2. We have also shown how these methods perform with regard to these metrics under low, medium, and high traffic flow levels. Qoordination has not been examined in this scenario, nor has any other form of coordination.

We have observed that under low traffic flow levels all methods' performances converge as the network size increases in many of the metrics. Although the reasons behind such a convergence are unclear we can see the opposite effect occur as the traffic flow levels increase. Under medium traffic flow levels the performances show no clear sign of convergence or divergence. Under high traffic flow levels the performances of the more static control methods diverge from those of the more dynamic control methods. The more dynamic traffic control methods, namely SAT and actuated control, perform similarly under high traffic flow levels because they are at maximum capacity and keep the phase lengths of their approaches that have traffic on them at maximum. These methods achieve little or no sense of coordination under high traffic flow levels. The more static control methods however do maintain some

coordination. Round Robin does not change its phase or cycle lengths so offset is never changed. It thus continues to allow the simultaneous progression enabled on all control methods at the start of the simulation i.e. all traffic light cycles start at the same time initially. We consider Q-Learning based control as somewhat of a static form of traffic control as it is slower to adapt to traffic than actuated control and SAT and this seems to allow its traffic controllers to keep their cycle lengths more similar to each other. Thus the simultaneous progression is again maintained throughout the experiment and the Q-Learning based control's performance excels because of it.

5.6.1.3 Pre-Coordinated Good / Bad

In order to have a benchmark for coordination in these experiments we set up a series of pre-coordinated simulations. In these simulations we set the offsets between each intersection to an optimum setting. We deemed this setting to be 15 seconds as this offset keeps the downstream intersections clear of traffic. In practice this is similar to pre-timed control where the offsets are set beforehand to an optimal setting. This of course has the drawback of needing to be calculated for each individual intersection, although in our case the network is a grid with road segments of equal length so optimal offsets of 15 seconds are uniform throughout the network. Another drawback is that the coordination will be set to move in a north easterly direction. If the traffic begins moving in the opposite direction i.e. a south westerly direction then the offsets would need to be set to -15 seconds. We have thus established coordination in both directions. North east coordination serves to show the effects of well-coordinated intersections. This could be looked on as a best case scenario. This is referred to as "Pre-Coordinated Good" in results shown in this thesis. South west coordination serves to show the effects of ill-coordinated intersections. This could be looked on as a possible worst case scenario. This is referred to as "Pre-Coordinated Bad" in results shown in this thesis.

5.6.1.4 Actuated Traffic Control

Although actuated control (see section 2.3.2.2) has the ability to rapidly adapt to changes in traffic flow it is traditionally deemed unsuitable for coordination. Before we progress into evaluating Qoordination on our chosen traffic control methods we wish to establish the truth of this. The reason for this is that actuated control requires decisions to be made roughly once every second. Round Robin need never make any decisions as its phase lengths, cycle length, and offset stay constant. SAT and Q-Learning based control, as well as Qoordination, require decisions to be made after an evaluation period of a cycle length or a set number of cycle lengths. Because actuated control's decision rate is so much higher than Qoordination's the two may have difficulty interfacing with each other. If we can establish that actuated control shows potential to coordinate actions between actuation based controllers then it

would be worth addressing these interface issues. If however actuated control does not show sufficient potential for coordination then the effort spent in addressing interface issues will have been wasted.

For our implementation of actuated control we set a minimum green time of the default phase duration i.e. 60 seconds, a maximum green time of 90 seconds, and an extension time of 6 seconds. The same phases and phase sequence are set up for actuated control as are for Round Robin, SAT, and Q-Learning based control, which are given in section 5.3. In our experiments phase skipping is not permitted.

Illustrated below are the effects of coordination on the metric of average vehicle waiting time for actuated control under low, medium, and high traffic flow levels.

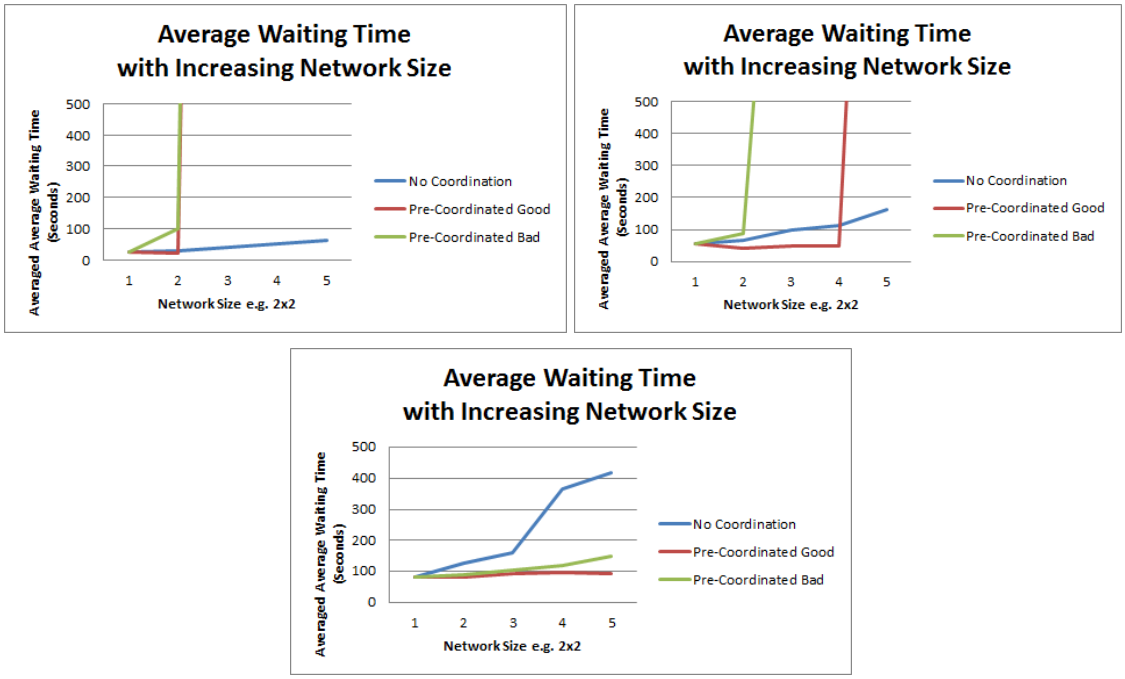

Figure 53 Round Robin waiting times with increasing network size, low/medium/high traffic flow levels

In the diagrams of Figure 53 we can see that with actuated control average vehicle waiting times tend to reach unacceptably high levels even with optimal coordination under medium and low traffic flow levels. The reason why this happens under medium and low traffic flow levels and not under high traffic flow levels is that under high traffic flow levels the queues are continuously full. Thus the phase lengths remain consistently at their maximum values and the cycle length remains consistent. Under medium traffic flow levels the phase lengths and thus the cycle length fluctuate making it more difficult to maintain coordination. Under low traffic flow levels the phase lengths are kept close to minimum levels resulting in such short cycle lengths that coordination becomes very quickly too difficult to maintain.

From these findings we can deduce that actuated control is indeed not suitable for coordination and we will thus not proceed with implementing a Qoordination interface. Actuated control will thus not feature in the remainder of the evaluation experiments.

5.6.1.5 Effects of Network Size and Coordination on Traffic Control Methods

We will now look at the effects of Coordination on the traffic control methods in increasing network sizes. It is thus at this point that we begin to evaluate Qoordination itself as opposed to the comparison of the traffic control as we have done so far. Our evaluation experiments show similar patterns in the results across the three different traffic control methods, Round Robin, SAT, and the Q-Learning based method. For this reason we include the results of only one method, namely SAT.

5.6.1.5.1 Results and Analysis

In this sub-section we present the results of this experiment scenario and analyze these results. We further analyze these results and discuss observed patterns and findings in sub-section 5.6.1.5.2.

5.6.1.5.1.1 Medium Traffic Flow Levels

We will begin by looking at the effects of coordination on SAT under medium traffic flow levels.

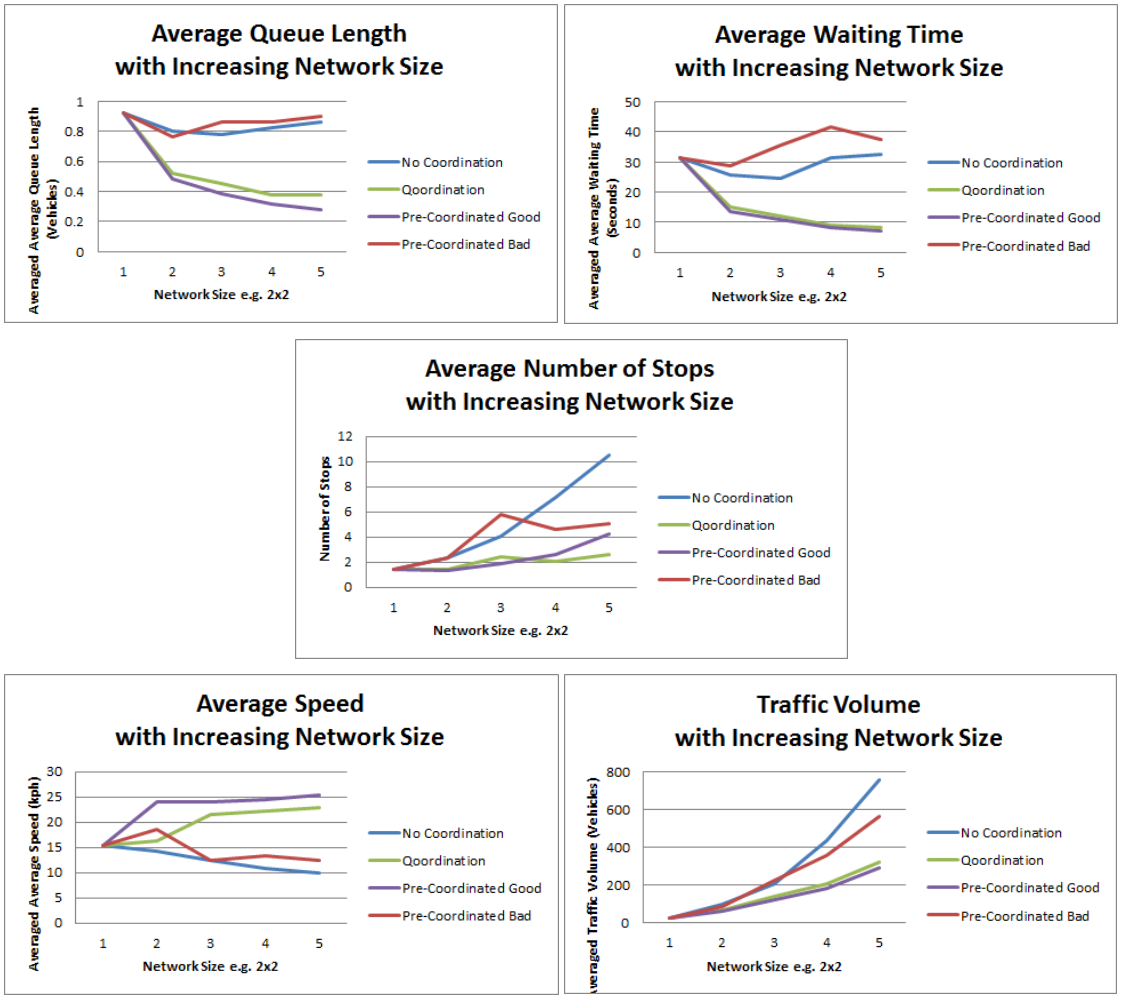

Figure 54 Effect of coordination on SAT with increasing network size, medium traffic flow

Figure 54 contains the first set of diagrams thus far in this thesis that specifically highlights the benefits of coordinating traffic light controllers within a network. This scenario specifically shows us that close to optimal coordination can be established using Qoordination. A significant difference between Qoordination and the pre-computed optimal coordination is that whereas the latter required manual setup and is not able to adapt to changing traffic flow directions or traffic flow levels the former learned on its own to establish coordination and is able to adapt to changing traffic flow directions or traffic flow levels. These diagrams are thus the most important that we have presented so far in this thesis.

In the first diagram of Figure 54 we can see that Qoordination reduces average queue lengths by 56% when compared to non-coordinated SAT. These improvement increases even further as network size continues to increase. These levels of reduced queue lengths are quite similar with optimum coordination levels i.e. Pre-Coordinated Good. These findings illustrate to us that Qoordination truly can learn nearly optimum offsets such that progressive signal systems can be established. We can also see in this diagram that the pre-coordinated bad based controllers performed only slightly worse than the non-coordinated controllers. This reason why this difference in performance is so small is that as a dynamic traffic control method SAT was able to compensate somewhat for the bad coordination. To prove this we include the following diagrams, which show that the performance difference between pre-coordinated bad based controllers and non-coordinated controllers is much higher under the same conditions for the more static traffic control methods i.e. Round Robin and Q-Learning based control.

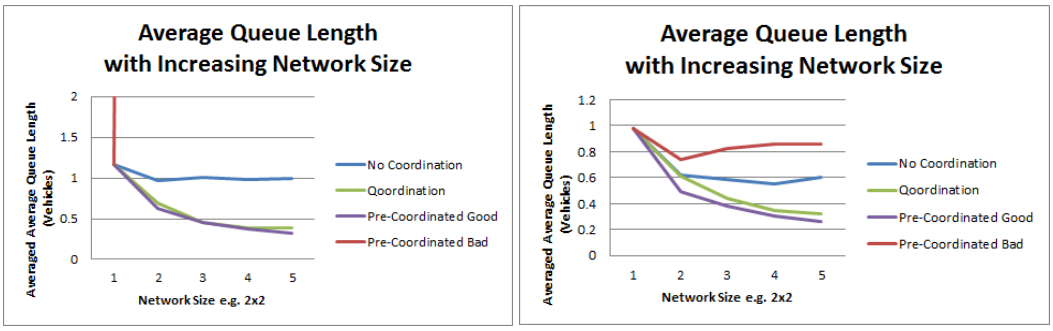

Figure 55 Effect of coordination on Round Robin and Q-Learning based control with increasing nw. size

In the second diagram of Figure 54 we observe that Qoordination reduces average vehicle waiting times by approximately 75% when compared to non-coordinated SAT by the time the network size has been increased to 5x5 intersections. Again these improvements continue to increase as network size increases. This is also consistent with optimal traffic coordination. This impressive increase in performance is even more reflective of the benefits of using Qoordination in larger traffic networks than those of queue lengths as these also take into account the amount of time the vehicles are waiting at the intersection.

With regards to the metric of average number of vehicle stops (the third diagram of Figure 54) we can see that the rate of increase of number of stops for Qoordination is more than significantly different than for non-coordinated control. In networks with no coordination implemented we can see that the average number of stops increases by approximately 620% as the network size increases from a single intersection up to a 5x5 network. Qoordinated networks however only increase by just under 80% over the same increase in network size. Thus Qoordinated networks are much more scalable than non-coordinated networks. Low number of stops is indicative of establishment of progressive signal systems. This proves that Qoordination does in fact achieve its design requirement Req1. We can also see that with regards to this metric Qoordination even outperformed what we considered to be an optimal solution, namely pre-coordinated good.

With regards to the metric of average vehicle speed, illustrated in the fourth diagram of Figure 54, we can see that Qoordination performs approximately 56% better than non-coordinated control. Thus vehicles traveling through a Qoordinated transport network will travel on average over twice as fast as those traveling through a non-coordinated network. This metric however has an upper limit of 50 kph as this is the maximum speed set on all roads within the simulated networks. To achieve 50 kph would mean that no vehicle ever needs to stop or slow down as it passes through the network, so an average vehicle speed of approximately 23 kph is not bad, especially taking into account that most vehicles have to stop to be formed into platoons on entering the network by the border intersections.

In the final diagram of Figure 54 we are shown that in a network that consists of a single intersection coordination is not possible and thus the network fills up just as much for all approaches. By the time network size reaches 5x5 the Qoordinated network contains on average 57% less traffic than the uncoordinated network. In comparing these two simulations visually the difference is remarkable as the uncoordinated network appears quite full while the Qoordinated network appears relatively empty. In reality this would lead to significant reductions in noise and air pollution.

5.6.1.5.1.2 Low Traffic Flow Levels

Let us now consider the effects of coordination on SAT under low traffic flow levels.

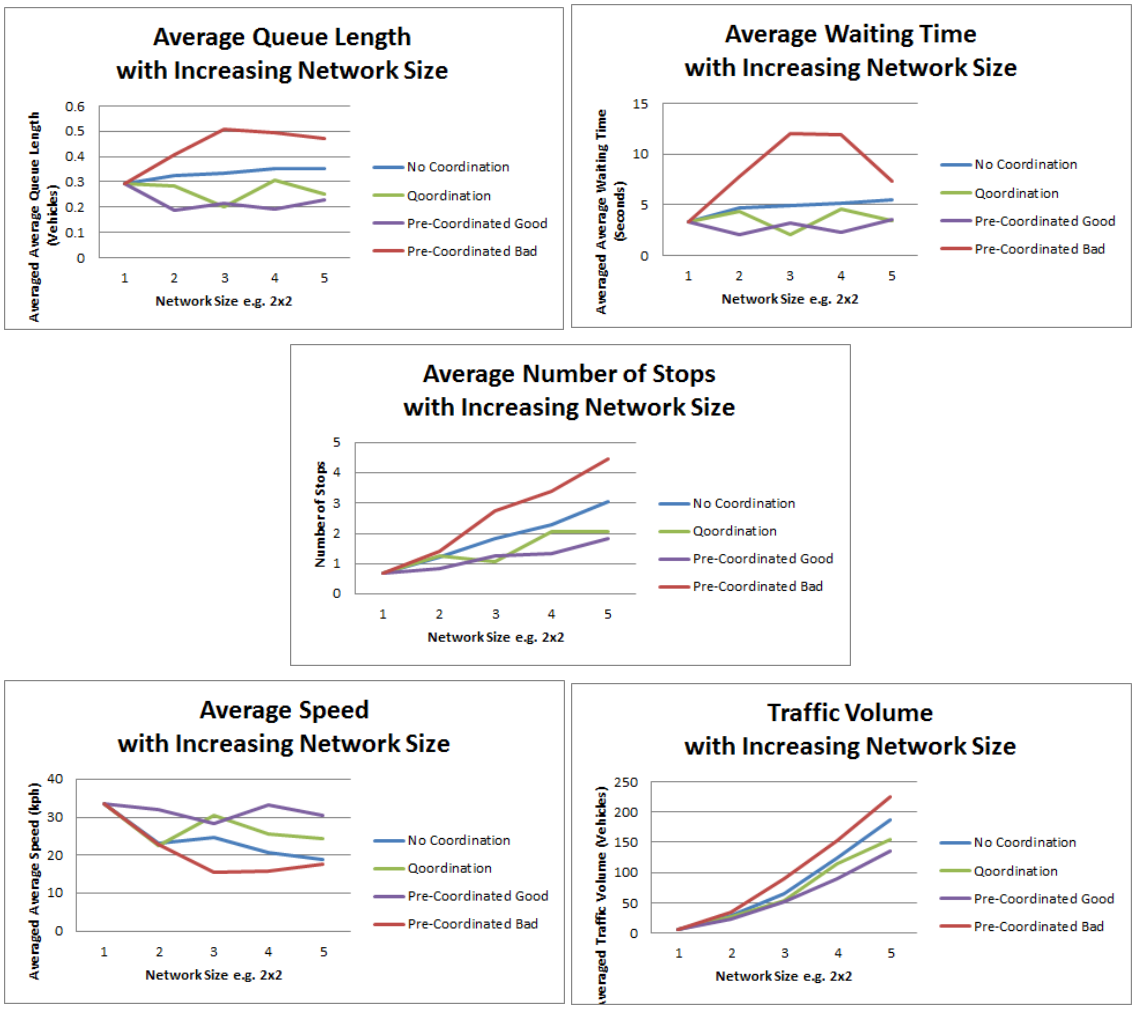

Figure 56 Effect of coordination on SAT with increasing network size, low traffic flow

Our first observation from the diagrams in Figure 56 is as we would expect based upon what we learned from the previous experiment scenario of section 5.6.1.2. That is, that under lower traffic flow levels the positive effects of coordination, and in turn Qoordination, are not as obvious as under higher traffic flow levels.

In the first diagram of Figure 56 we can see that queue lengths are reduced in 5x5 transport networks by approximately 30% when implementing Qoordination as opposed to when no coordination is established. These reductions in queue length fluctuate more so than those under medium traffic flow levels for reasons discussed in section 5.6.1.1.1.2. We can also see in this diagram that under low traffic flow levels pre-coordinated bad based intersection controllers appear to perform significantly worse than non-coordinated intersection controllers. This is an interesting discovery and when considered in this scenario alone its root cause is not apparent. When compared to the results of the medium and high traffic flow level scenarios however a pattern emerges. The higher the traffic flow the better pre-coordinated bad

based controllers perform, as opposed to non-coordinated controllers. The reason for this, we speculate, is that because pre-coordinated bad control maintains the offset between intersections at -15 seconds it does not let them drift further out of sync. Short cycle lengths cannot drift too far out of sync, resulting in a larger percentage of the time that uncoordinated signal controllers have an offset between 0 and -15 or even between 0 and +15. Thus because uncoordinated controller offsets can fluctuate they are often more in sync with their neighbors under low traffic flow levels than pre-coordinated bad controllers. Under high traffic flow levels this fluctuation in offsets is a drawback as the cycle lengths are much larger and the controllers spend the vast amount of time less in sync with their neighbors than pre-coordinated bad based controllers i.e. with offsets < -15 or > + 15.

In the second diagram of Figure 56 we are shown that in networks of size 5x5 intersections Qoordination leads to a reduction in average waiting times of approximately 38% when compared to non-coordinated SAT. This is an increased improvement over that of the queue length metric due to the fact that it also takes into account the amount of time that the vehicles spend in the queue. Average waiting time is thus somewhat more representative of the real improvement in traffic flow in the network than the queue length metric.

The third diagram of Figure 56 shows us that the average number of vehicle stops for Qoordination stays somewhat similar to optimal coordination, though both fluctuate slightly for reasons discussed earlier. Here there is a reduction of approximately 32% in average number of stops when using Qoordination as opposed to where no coordination is used. This figure is roughly in line with the first two diagrams of Figure 56.

Under low traffic flow levels the average speed seems to be significantly higher for pre-coordinated good based controllers than for Qoordination based controllers. Qoordination still however manages to have an increase in average speed of approximately 22% over non-coordinated intersections within the 5x5 network. The reason for a smaller level of improvement here is that there is an upper speed limit set of 50 kph. With such short cycle lengths average speeds are kept higher, even in uncoordinated control.

With regards to the traffic volume metric, as shown in the final diagram of Figure 56, Qoordination improves performance over non-coordinated SAT by approximately 17% in the 5x5 network. Again, this level of increase is not as high as in the other scenarios due to the fact that, as we have now observed, coordination itself improves performance relative to the level of traffic flow, which in this scenario is low.

5.6.1.5.1.3 High Traffic Flow Levels

Let us now look at the effects of coordination under high traffic flow levels.

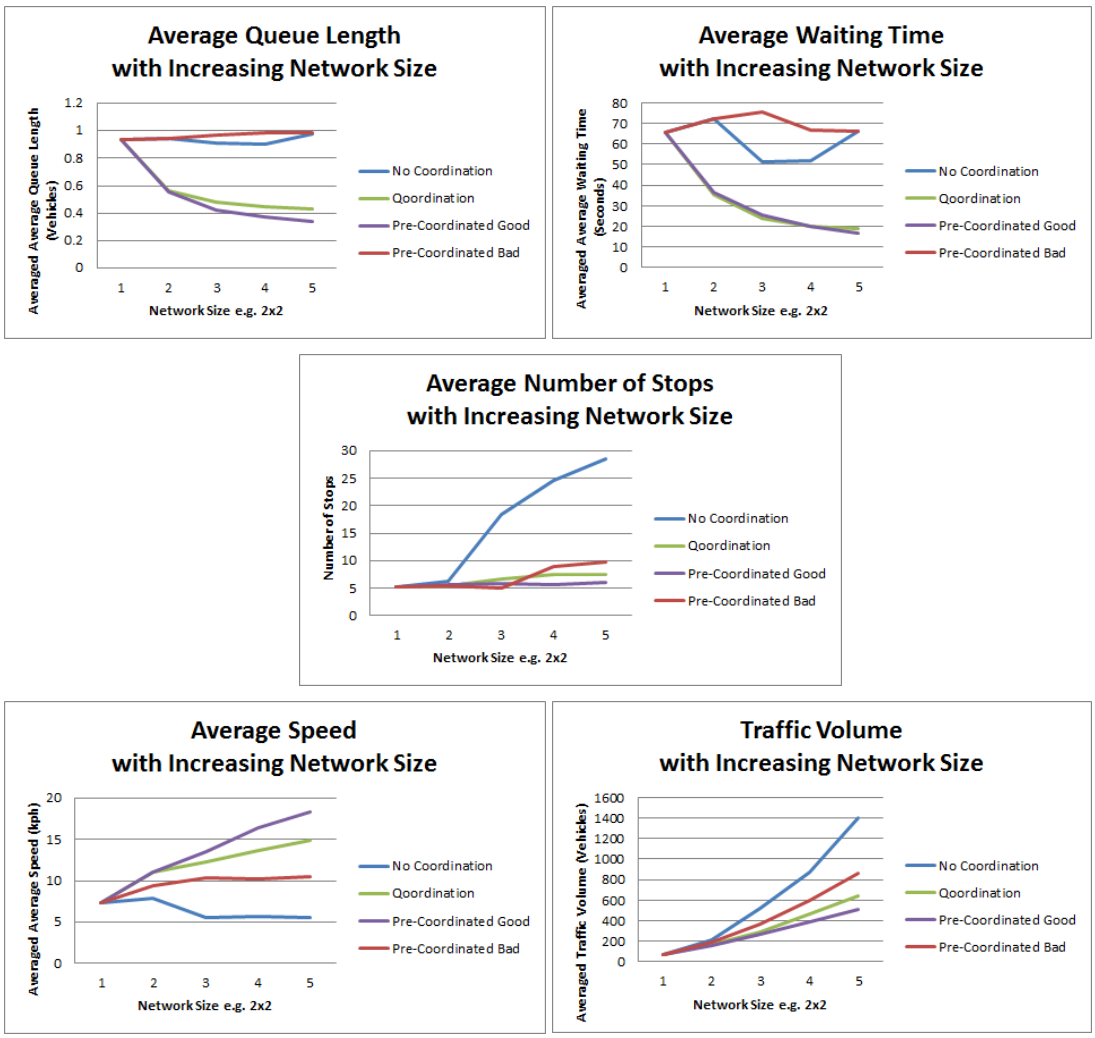

Figure 57 Effect of coordination on SAT with increasing network size, high traffic flow

In the first diagram of Figure 57 we can see that Qoordination reduces queue lengths by 55% when compared to non-coordinated control under high traffic flow levels. This comes as somewhat of a surprise as this value is slightly lower than that obtained under medium traffic flow levels. We had theorized that the positive effects of coordination increase not only as the network size increases but also as the traffic flow levels increase. Although this theory may still be valid we must append to it the possibility of there being a threshold traffic flow level at which the positive effects of coordination stop increasing. Thus our theory now is that the positive effects of coordination increase not only as the network size increases but also as the traffic flow levels increase towards a certain maximum threshold value. We will proceed with giving the results obtained in this scenario simulation and see if the remaining metric results confirm this theory.

In the second diagram of Figure 57 we observe that Qoordination waiting times are decreased by 72%% when compared to non-coordinated control. This compares to the result of 75% reduction in waiting times obtained under medium traffic control. Thus our appended theory of a maximum traffic flow level up to which Qoordination increases performance is confirmed with regards to the average waiting time metric.

The average number of vehicle stops for Qoordination, as can be seen in the third diagram of Figure 57, is significantly lower than for non-coordinated controllers. 74% lower, which compares to the 75% reduction achieved under medium traffic flow levels, again confirming our appended theory. We can see in each of these diagrams that Qoordination achieves results very similar to those achieved by pre-coordinated good based controllers. This again confirms that Qoordination learns near optimal coordination.

In the fourth diagram of Figure 57 we can see an increase of 63% in average vehicle speed by implementing Qoordination as opposed to not implementing coordination. This 63% compares to the 56% achieved under medium traffic flow levels. With regards to this metric Qoordination has thus continued to improve performance as traffic flow levels increase.

The final diagram of Figure 57 shows that Qoordination reduces traffic volumes by 54% in a 5x5 network when compared to non-coordinated SAT.

We will now discuss and analyze the results obtained in our experiments of this evaluation scenario.

5.6.1.5.2 Concluding Analysis

This experiment scenario has highlighted the effects of coordination on the SAT traffic control method in increasing network sizes and under different traffic flow levels. Through this scenario we have shown that Qoordination reduces queue lengths and average vehicle waiting times similarly to optimal coordination i.e. pre-coordinated good control. We have found that the positive effects of coordination increase both as the size of the network increases and as the traffic flow levels increase towards a certain maximum threshold value. In the following diagram we graph the increase in performance obtained by Qoordination as opposed to non-coordination under low, medium, and high traffic flow levels, for all metrics considered.

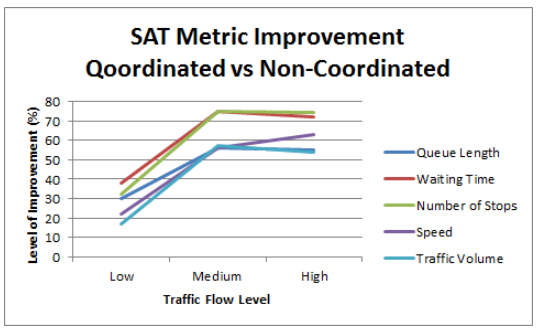

Figure 58 SAT metric improvement in a 5x5 network in low, medium, high traffic flow

This diagram clearly shows that Qoordination improves performance over non-coordination with regards to all metrics. The level to which the performance is improved increases significantly between low and medium traffic flow levels. Between medium to high levels however this rate of increased performance improvement levels off and is slightly reduced. This confirms our theory that there is a maximum traffic flow level up to which Qoordination increases performance. This maximum value however may not be the same for each of the three traffic control methods i.e. SAT, Round Robin, and Q-Learning based control. The following diagrams illustrate this point.

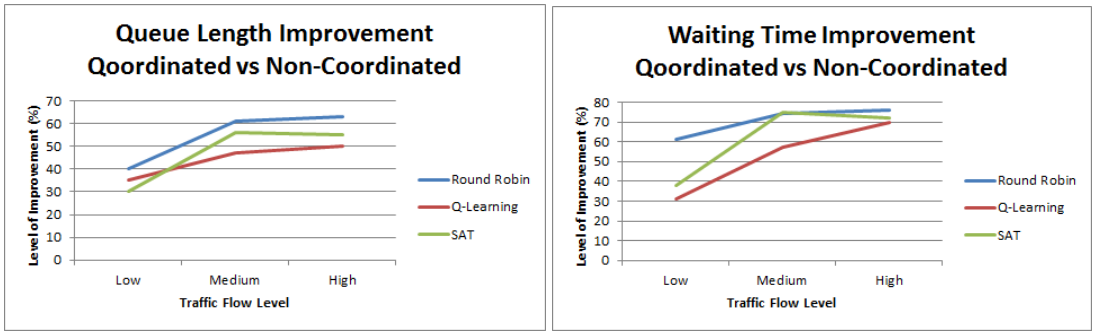

Figure 59 Control method improvements in a 5x5 network in low, medium, high traffic flow

These diagrams show that the use of Qoordination results in improved queue lengths and waiting times in a 5x5 network in low, medium, and high traffic flow levels, for SAT, Round Robin, and Q-Learning based control. They further show that for Round Robin and Q-Learning based control the not only does the level to which the performance is improved increases significantly between low and medium traffic flow levels but for Round Robin and Q-Learning based control this level of improvement continues to increase between medium and high traffic flow levels. With regard to our theory this would indicate that for these two traffic control methods the maximum level of traffic flow up to which Qoordination improves performance has not yet been reached. This could highlight that this maximum value is higher for the more static traffic control methods.

5.6.1.6 Conclusion

In this first experiment we have shown a number of things. In the first scenario we have established how well the traffic control methods that we have chosen to incorporate into this evaluation perform in a transport network that single intersection. From this scenario we learned that for a single intersection Round Robin performs consistently worse than the other control methods while SAT performs best. In low traffic flow levels the metric readings fluctuated significantly due to the fact that the queue lengths vary significantly between empty and full at any one time. Queue lengths in medium traffic flow levels on the other hand tend toward being full a lot of the time, which has a smoothing effect on the different

metrics, as described in section 5.6.1.1.1.2. Under high traffic flow levels the control methods performed more similarly due to the fact that they were constantly at full capacity.

In the second scenario we have extended the single intersection network a number of times so that we can observe how each of the control methods performs in networks of increasing size. These networks range from the single intersection scenario up to a 5x5 intersection network. In this particular scenario no coordination is taken into account so each intersection makes modifications to its timing plan without regard for the other intersection controllers in the network. From this scenario we have learned that as the network size increases the rate of change in performance of more static approaches to traffic control i.e. those whose cycle lengths remain similar across the intersections within the network, is much improved when compared to the rate of change in performance of more dynamic control approaches. This shows early signs that maintaining similar cycle lengths across intersection controllers within the network, which enables coordination, leads to better performance.

We then showed that the actuated control method is not well suited to coordination and so will not be considered any further in our experiments.

In our third scenario we extend the previous scenario so as to include pre-timed coordination as well as Qoordination. This scenario thus highlights the effects of coordination on traffic control methods in increasing network sizes and under different traffic flow levels. Through this experiment we show that Qoordination reduces queue lengths and average vehicle waiting times similarly to optimal coordination. We found that the positive effects of coordination increase both as the size of the network increases and as the traffic flow levels increase towards a certain maximum threshold value. This value is not the same for the different traffic control methods and seems to be higher for the more static control methods. In our largest network in this experiment i.e. a 5x5 intersection network, and under high traffic flow levels we have seen a reduction in average waiting times of 72% as a direct result of Qoordination. These findings highlight Qoordination's ability to form progressive signal systems and thus directly address research objectives 1, 4, and 5 as well as the first research question that we defined in section 1.3.

In conclusion, this first experiment confirms that Qoordination can learn to coordinate traffic flow throughout a network in that results in close to optimum performance. Coordination in turn has been proven to increase network performance, with regards to a number of metrics. This increase in performance grows as network size grows and as traffic flow levels increase up to a maximum value. This experiment however only shows the benefits of coordination and Qoordination in networks whose traffic flows in north and east bound directions simultaneously. It further only shows these benefits in static traffic flow levels. Within the remaining experiments we will show the benefits of using Qoordination in networks of changing direction and traffic flow levels.

5.6.2 Round Robin Justification

The results of experiment 1 show that the patterns of the effects of Qoordination on Round Robin, SAT, and Q-Learning based control show similar patterns. Due to this fact we have decided that we are going to use only one of these control methods in the remaining experiments. This greatly reduces the number of experiments that need to be run as well as the number of diagrams to be analyzed in this thesis. The question remains as to which control method to use. The purpose of this evaluation to establish that Qoordination achieves the evaluation objectives set out in section 5.1, not to compare how well Qoordinated SAT performs against Qoordinated Round Robin. Neither is it the purpose of this evaluation to find the best control method to use in conjunction with Qoordination. We can however observe from Figure 59 that Qoordination has greater effects on Round Robin than it does on SAT and on Q-Learning based control. As we wish to highlight Qoordination's effects on a control method Round Robin this makes Round Robin a good choice. Additionally, being a more static control method, Round Robin based Qoordination takes less time and fewer simulations to train, reducing the workload required to perform this evaluation. The remainder of the experiments performed for this evaluation will thus only feature the Round Robin traffic control method unless otherwise stated.

5.6.3 Experiment 2 – Changing Traffic Flow Direction

The purpose of this experiment is to show that Qoordination can adapt to changing traffic flow directions. In this experiment we change traffic flow direction mid simulation. For the first half of the simulation the traffic flows through the network in a north east direction and in the second half of the simulation it flows south west. Throughout the duration of the simulation however the traffic flow level is kept constant. The purpose of this experiment is to meet objective 2 i.e. assess how well Qoordination can adapt to changing traffic flow directions. This experiment is performed in a network of 2x2 intersections. All roads in this experiment are one way roads. No left or right turns are permitted. Traffic flow levels in this experiment are kept at a steady medium traffic flow level, permitting 650 vehicles per hour through each of roads that allow traffic to enter the network. The simulation has a duration of 21,600 seconds i.e. 6 hours.

5.6.3.1 Results and Analysis

To highlight the results obtained in this experiment from Round Robin based Qoordination we compare it to non-coordinated Round Robin and also the optimally coordinated Round Robin described in the previous experiment i.e. Pre-Coordinated Good. Again we draw the reader's attention to the fact that this optimally coordinated method is static and cannot adapt to changing traffic flow directions. It is optimized for traffic flowing in a north east direction, which is the direction of traffic flow for the first half of the simulation. From the previous experiment we were able to observe that the two main

evaluation categories that Qoordination has an effect on are average queue length and average waiting time. Although Qoordination does not have a negative effect on any of the other categories the positive effect that it does have are not as dramatic as with the first two. For this reason the only two evaluation categories that we will cover here are average queue length and average vehicle waiting time.

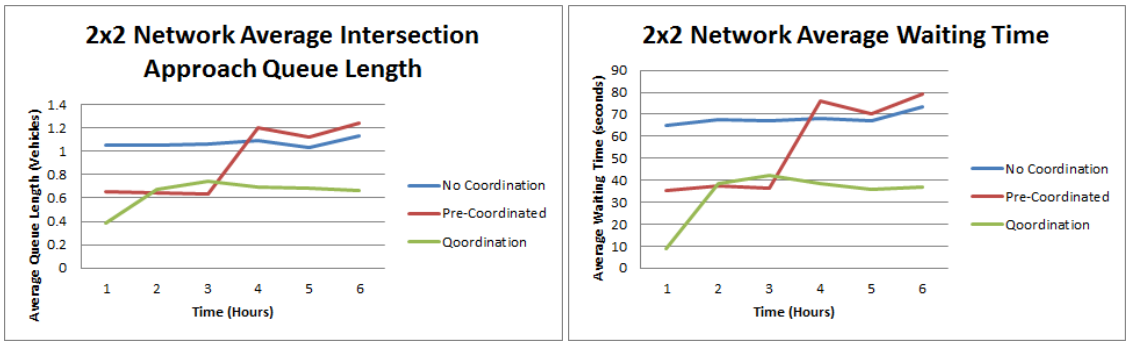

Figure 60 Queue lengths and waiting times in reversal of traffic flow direction

The reader's attention is drawn to the fact that in Figure 60 we are again observing the passing of time since the start of the simulation and not increases in network size. From the results shown in these diagrams we can see that when no coordination is applied i.e. when a simultaneous progressive signal system is in place, the reversal of the traffic flow direction has no effect on the performance of the network. This result is expected as all traffic lights in the networks change at the same time and thus traffic progression in any one direction will be impeded just as much as traffic progression in any other direction. As the optimally Pre-Coordinated Good method is only optimized for the direction of traffic flow in the first half of the experiment there is a dramatic change in performance once the traffic flow direction changes. Both average queue length and average vehicle waiting time maintain a steady level of approximately 65% of those of the non-coordinated approach throughout the first half of the experiment. Once the direction changes however both of these metric values shoot up to being approximately 7% higher than those of the non-coordinated approach. The Pre-Coordinated Good method thus essentially becomes Pre-Coordinated Bad once the direction of traffic flow is reversed.

We can observe however that Qoordination is able to adapt to the reversal of traffic flow direction. Qoordination average queue length and average vehicle waiting time both raise ever so slightly when the direction of traffic flow changes but they immediately return to their previous performance levels thereafter. This effect is as a direct result of the complete reversal of the progressive signal system that Qoordination establishes within the simulated network. Qoordination is thus shown to be able to adapt to the change in traffic flow direction. The reason why the immediate increase in queue length and average waiting time after the change in direction is difficult to see in these diagrams is because the time taken for Qoordination to reverse the progressive signal system established in this small 2x2 network is only a matter of minutes. This is then smoothed out in the graph due to the fact that each point on the graph is an

average of values observed over the course of an hour of the simulation. In larger networks this reversal would become more obvious in the diagrams as it would take longer to reverse the larger progressive signal systems.

5.6.4 Experiment 3 – Changing Traffic Flow Levels

The purpose of this experiment is to show that Qoordination is able to adapt to changes in traffic flow levels. In this experiment the traffic flow levels are varied from medium to high mid simulation. This satisfies evaluation objective 2. As opposed to using Round Robin traffic control method we decided to use SAT instead. This is because of SAT's more dynamic nature which makes it more able to adapt to changes in traffic flow levels. It is thus up to SAT to adjust its phase lengths so as to adapt to the changes in traffic flow levels. Qoordination's task is to maintain coordination among the SAT controlled intersections throughout the change. In this experiment traffic flows consistently from the west of the network to the east and from the south of the network to the north. All roads in use are one way roads. No left or right turns are permitted. Medium traffic flow levels permit 650 vehicles per hour and heavy traffic flow levels permit 950 vehicles per hour through each of the north bound and east bound roads i.e. each of the roads that allow traffic to enter the network. Each simulation has a duration of 21,600 seconds i.e. 6 hours. This experiment is conducted within a 3x3 intersection network.

5.6.4.1 Results and Analysis

The evaluation metrics that we use to highlight these evaluation results are average vehicle waiting time, average vehicle number of stops, and network traffic volume as these metrics best highlighted the results.

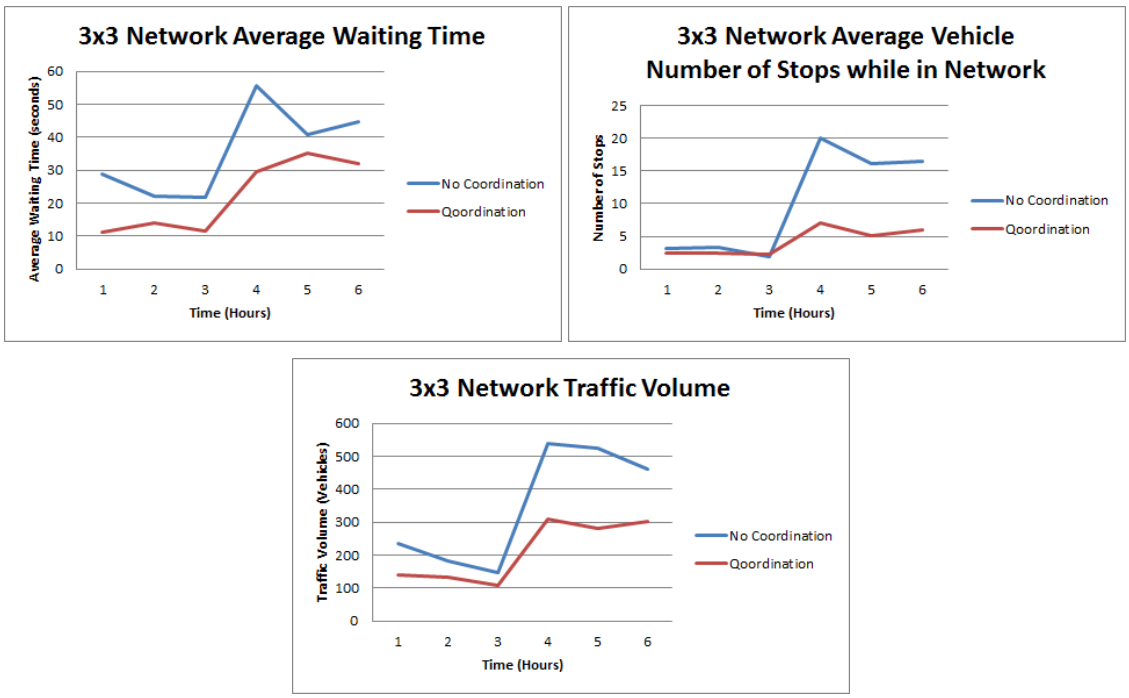

Figure 61 Waiting times, number of stops, and traffic volume in fluctuation of traffic flow levels

From the above results we can observe that the increase in traffic flow levels, which occurred half way through the experiment, leads to similarly higher vehicle waiting times for both Qoordinated and non-coordinated approaches. This effect however is not present in the category of average number of vehicle stops. Whereas the average number of vehicle stops were roughly equal before the change in traffic flow levels they are not afterwards. After the change Qoordination achieves an average number of stops three times as low as those that are achieved where no coordination is used. This is a significant difference. Traffic volume levels however also maintain their appropriate ratios after the change in traffic flow levels. These results insinuate that Qoordination is able to adapt appropriately to changes in traffic flow levels.

One further aspect of this experiment was the results obtained in the category of throughput.

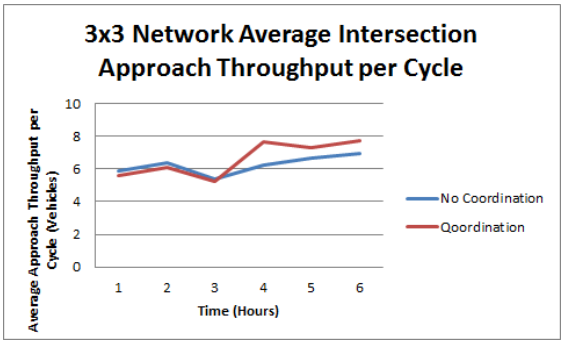

Figure 62 Average throughput in fluctuation of traffic flow levels

Whereas we expected no significant change in throughput to come as a result of the change in traffic flow levels we found that after the change a significant change of roughly one additional vehicle being put through each approach on each intersection per cycle resulted. This came as a surprise as we had not noted significant increases in vehicle throughput using Qoordination under higher traffic flow levels in experiment 1.

5.6.5 Experiment 4 – Reversing a Single Progressive Signal System

The main purpose of this experiment is to establish a progressive signal system along a main traffic corridor that does not take a direct route from one side of the network to the other. In this experiment Round Robin control is implemented on the intersections in a 2x2 network. In this experiment the main traffic corridor in question sources from the west of the network, passes straight through the first intersection, takes two consecutive right turns, then passes straight through the next intersection to exit the network again on the west. Mid-way through this experiment the direction of traffic flow is reversed. This experiment thus achieves both research objective 1 and 2. In its initial direction the flow of traffic is more limited than in the reversed direction. The reason for this is because the traffic control method being implemented in this experiment is that of Round Robin. With the Round Robin method the dedicated right turns have only a 30 second phase length. Left turns however can be taken during a 30 second phase as well as the main 60 second phase. This will mean that traffic during the first half of the experiment will likely get congested at the second intersection that the vehicles meet after they enter the network i.e. the top right intersection. Traffic flow levels of 150 vehicles per hour are maintained throughout this experiment so that congestion on the downward bend does not cause an issue. In practice SAT would likely be a more appropriate traffic control method as it would be able to adapt the phase lengths of the dedicated right turns to accommodate higher traffic flow levels. The traffic corridor used in this experiment is shown below:

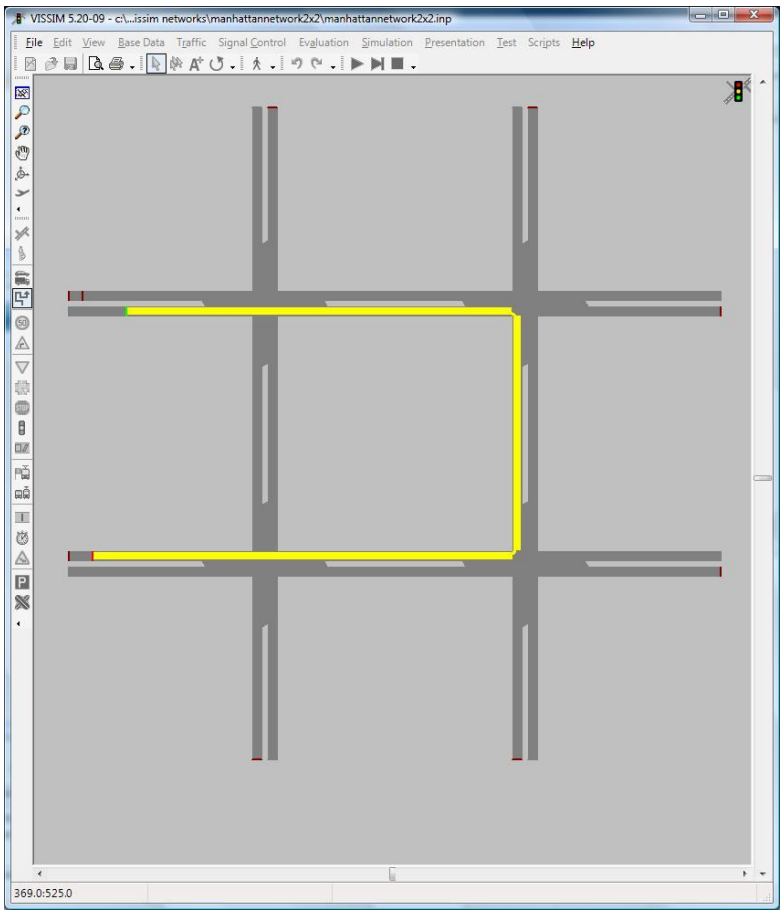

Figure 63 Experiment 4 main traffic corridor

5.6.5.1 Results and Analysis

The evaluation metrics that we will look at in this experiment are average number of vehicle stops, average queue length, and average vehicle waiting time.

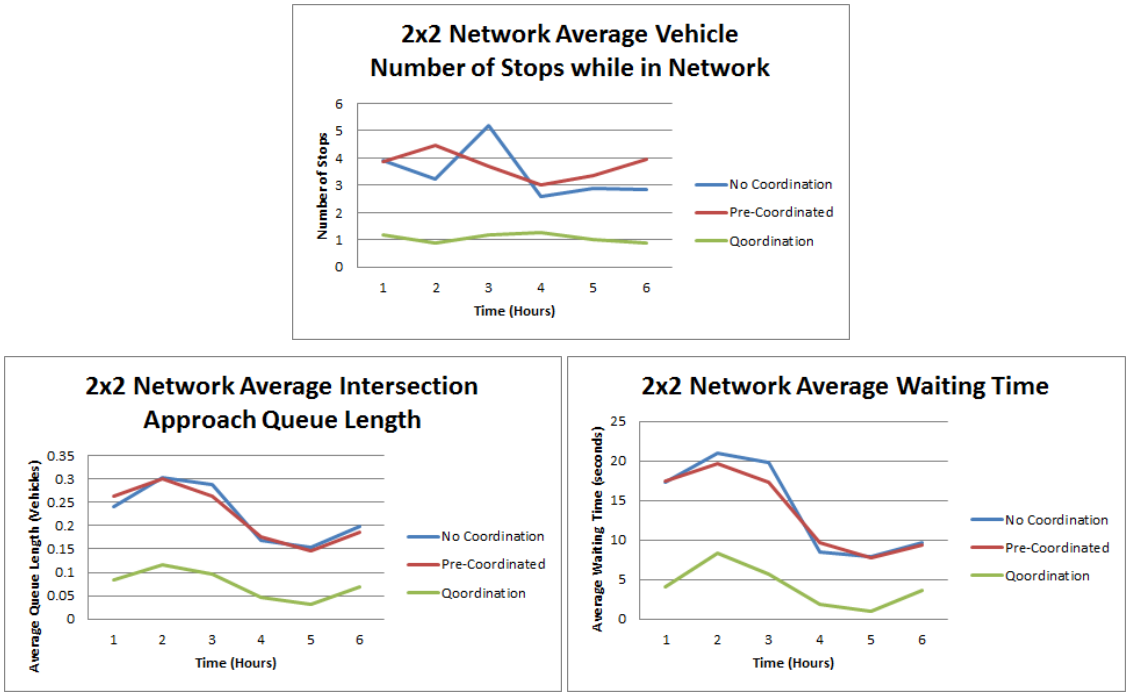

Figure 64 Experiment 4 average number of vehicle stops, queue lengths, and vehicle waiting times

We can observe from the first of these diagrams a slight fluctuation in the Qoordination average number of vehicle stops. This value starts slightly above a value of 1 as it takes a little while for the progressive signal system to be established. When it is established the number of vehicle stops drops below 1. With the change in traffic flow direction the number of stops is increased while the direction of the progressive signal system is reversed. We noted that it took approximately the equivalent of 50 minutes real time for this complete reversal of the progressive signal system. Once the progressive signal system has been reversed the average number of vehicle stops again drops down to a value of 1 or under. Pre-coordinated traffic control has the coordination moving in a north east direction, thus it is not optimal for this scenario. Calculating and implementing a pre-coordinated set of offsets for this scenario would have been a time consuming process. This is because once turns are taken into account optimal offsets are no longer consistently 15 seconds as the phases that feature the turns occur at different points throughout the cycle. The pre-coordinated and non-coordinated approaches achieve average number of vehicle stops up to four or five times those achieved by Qoordination. After the change in traffic flow direction pre-coordinated and non-coordinated achieve lower average vehicle stops than before the change. The reason for this is the limitations to this corridor due to the Round Robin method of control that we explained earlier on in this section. The same phenomenon can be seen for all forms of coordination or lack thereof on the two remaining evaluation metrics. The reader may note however that in the second half of the experiment the lowest average vehicle waiting time achieved by Qoordination is an impressive 1 second,

while the lowest value achieved by pre-coordination and non-coordination is approximately 8 seconds. The highest average vehicle waiting time achieved by Qoordination in the first half of the simulation is 8.3 seconds, while the highest value achieved by the other methods is approximately 21 seconds.

5.6.6 Experiment 5 – Complex Progressive Signal System

In this experiment we wish to illustrate the effects of the establishment of progressive signal systems with a much more complex traffic path. In this experiment we use a 3x3 intersection network that maintains a steady traffic flow level of 150 vehicles per hour across Round Robin intersections. The path taken in this experiment is shown below:

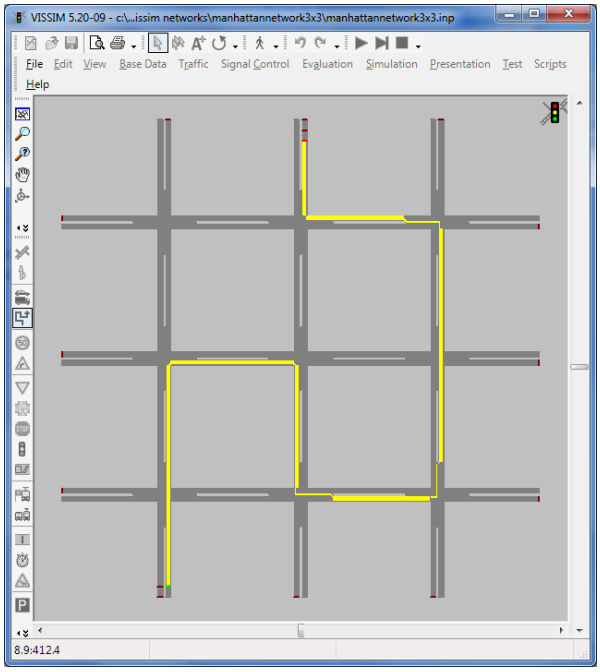

Figure 65 Experiment 5 main traffic corridor

The purpose of this experiment is to show that the benefits of Qoordination increase as the complexity of the main traffic path through the transport network increases. This also confirms evaluation objective 1.

5.6.6.1 Results and Analysis

Let us now consider the following set of evaluation results:

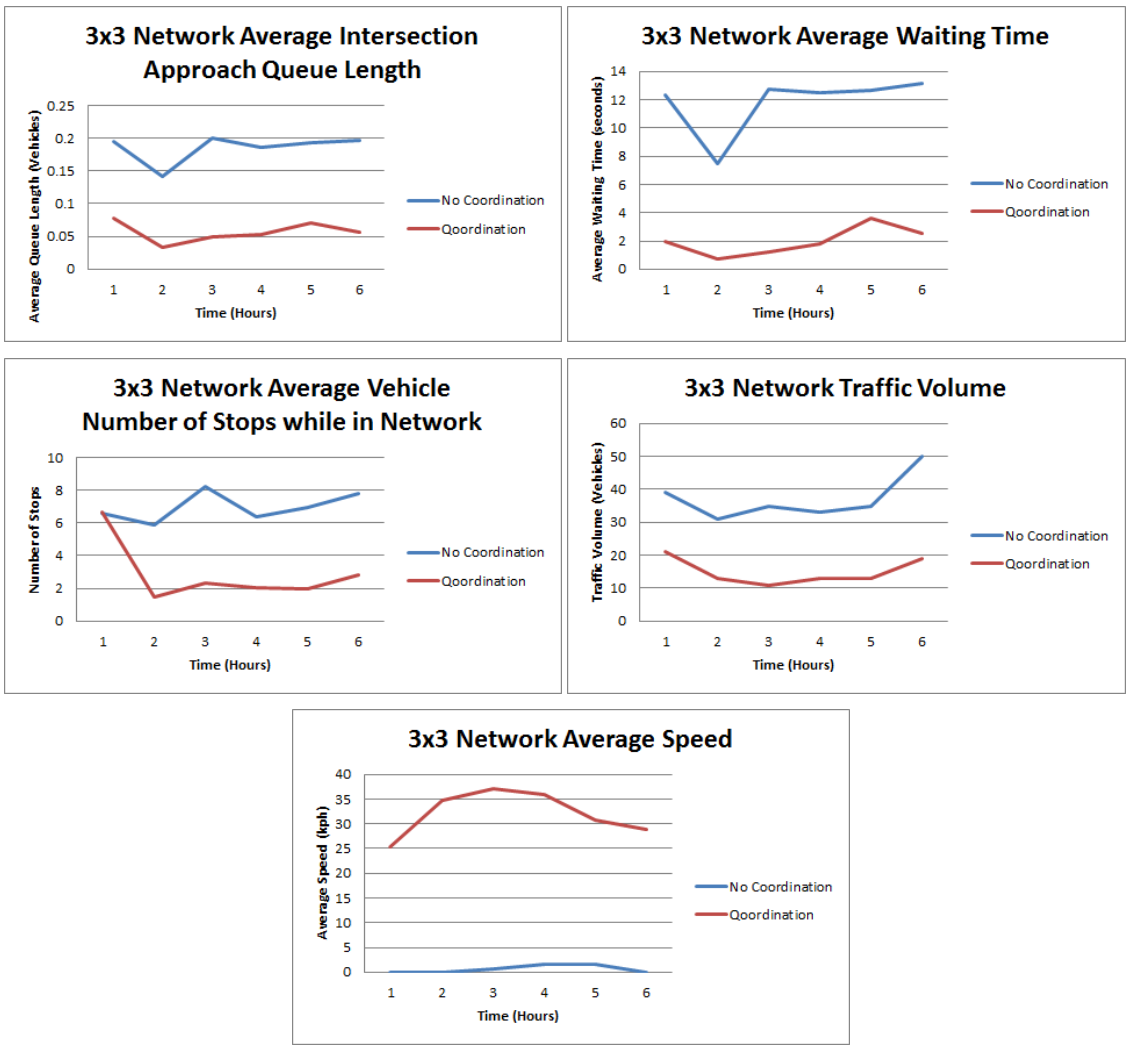

Figure 66 Experiment 5 evaluation results for complex traffic corridor path

Throughout the evaluation queue lengths are kept between roughly two thirds or three quarters less for Qoordinated intersections as for non-coordinated intersections. This results in an average reduction of queue lengths of 70% in this scenario due directly to the use of Qoordination. Waiting times are reduced by an average of approximately 83%, number of vehicle stops by 58%, and total traffic volume by 60%. Average vehicle speeds through the network are increased by almost 50 times those of non-coordinated intersections due directly to the application of Qoordination to this scenario. These phenomenal increases in evaluation results would strongly insinuate that the benefits of using Qoordination in a transport network increase dramatically as the level of complexity of the main traffic corridor increases.

5.6.7 Conclusion

In this section we have evaluated Qoordination through experimentation. The experiments that we have performed as part of this evaluation establish how well Qoordination addresses the requirements defined in section 3.1. The ultimate goal of these requirements is to address our research questions that were defined in section 1.3. To address these requirements we have set out clear evaluation objectives in section 5.1. Each experiment performed in this section has been performed with the aim of meeting one or more of these objectives.

Experiment 1 was designed to meet objective 4 i.e. analysis of how well Qoordination scales with increasing network size. Through this experiment we discovered that that the positive effects of coordination increase not only as the network size increases but also as the traffic flow levels increase towards a certain maximum threshold value. Qoordination has been shown to achieve results similar to optimally pre-timed control, proving that it can learn to coordinate traffic timing plans across intersections in the network in an optimal fashion. We have shown that when traffic control methods are integrated with Qoordination they scale significantly better as the network size increases than if no coordination were taken into account. We have also shown that as traffic flow levels increase up to a certain maximum value for each control method so too do the improvements in performance achieved by implementing Qoordination.

Experiment 1 was designed to also meet objective 5 i.e. assess the effects of Qoordination of various methods of traffic control. We have shown that actuated control is unsuitable for coordination in general. It was thus not considered in any further experiments. We have shown that the three other control methods to which Qoordination has been applied have shown significant increases in performance (see Figure 59). With regards to queue lengths Qoordination has shown to improve Round Robin by 61%, Q-Learning based control by 47%, and SAT by 56% under medium traffic flow levels when compared to these same control methods performance when no coordination is taken into account. With regards to average vehicle waiting time Qoordination has further shown to improve Round Robin by 74%, Q-Learning based control by 57%, and SAT by 75% under the same conditions. The general trend that we can see is that it leads to improvements in all three control methods, with slightly higher improvements for Round Robin and slightly less improvements for Q-Learning based control.

Experiment 2 was designed to meet objective 2 i.e. assess how well Qoordination can adapt to changing traffic flow directions. In this experiment we completely reversed the traffic flow direction mid simulation. Qoordination was able to adapt to this change so rapidly that it was even difficult to perceive a drop in performance in the graphed results. A statically pre-coordinated approach that was used as a benchmark however was affected dramatically by the change as it was unable to adapt. Through the results obtained from this experiment we can see that Qoordination is well capable of adapting to changes in traffic flow direction.

Experiment 3 was designed to also meet objective 2 by assessing how well Qoordination can adapt to changing traffic flow levels. We were able to show through this experiment that as traffic flow levels change the traffic control method e.g. SAT is able to optimize its phase lengths so as to adapt to these changes. This experiment shows that Qoordination is able to keep the intersections coordinated throughout this change.

Experiment 4 was designed to meet objective 1 by assessing how well Qoordination agents can coordinate their actions and establish progressive signal systems throughout the main traffic corridors of the transport network. In this experiment we create a main flow of traffic that does not flow directly through the network in a straight course, but that turns and then exits the network from the same direction in which it had entered the network. Not only was Qoordination able to automatically establish a progressive signal system along this main corridor, but also when we reverse the direction of traffic flow mid simulation Qoordination dynamically adapted to this change and reversed the direction of the progressive signal system. This experiment thus also meets objective 2.

In the final experiment we again focused on meeting objective 1 but as opposed to the previous experiment this one pushes Qoordination further by giving the main traffic corridor flowing through a larger network an intricate path that takes both left and right turns as it meanders through the network. Qoordination again automatically established a progressive signal system along the main traffic corridor which resulted in an average reduction of queue lengths by approximately 70% and a reduction in average vehicle waiting time a by approximately 83%.

With these experiments concluded only objective 3 remains to be met. This will thus be addressed next.

5.7 Multi-Layer Hashing Utility Function Analysis

In this section we analyze the inner workings of the MLH utility function to ensure that it is working properly. Throughout this analysis we look at the MLH utility functions of Qoordinated intersection agents within a 2x2 transport network. The reason for this choice in network size is because each intersection agent in the 2x2 network has a total of 2 neighboring intersections. As Qoordination agents maintain one state variable per adjacent intersection i.e. for its offset, this will lead to two dimensional state spaces. Two dimensional state spaces can be visualized quite efficiently and so patterns within these state spaces can be easier for us to confirm visually than the four dimensional state spaces of central intersections in 3x3 networks. Traffic flow in this network flows from south to north and from west to east. No turns are permitted. Round Robin traffic control is used in this analysis. The network in question is illustrated below with each intersection given its assigned number:

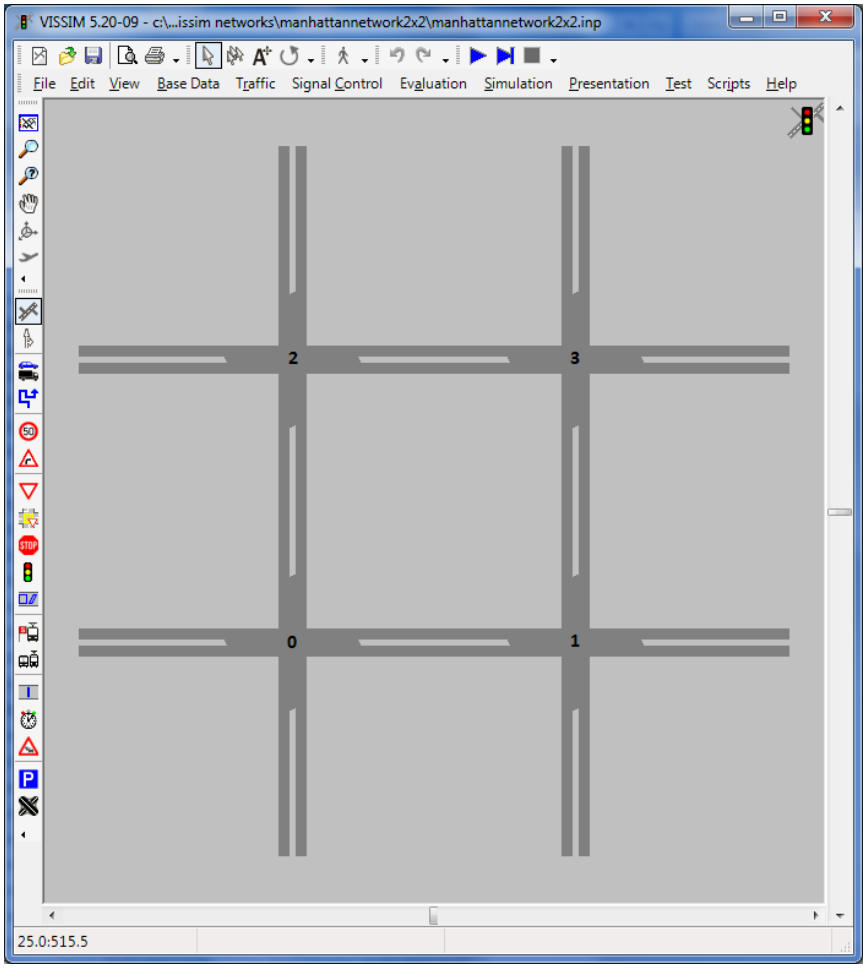

Figure 67 2x2 network for use in MLH utility function analysis

As described in Chapter 4, on initialization each agent performs each of its actions a specified number of times so as to get a grounding in their understandings of the effects of their actions. In this case this specified number is twenty. These actions are taken while no other relevant agents are taking any actions, thus reducing the dynamics of the observable effects. We begin this analysis by looking at an MLH's state space with regards to exploration during this initialization process. We then look at the state space of each of the agents immediately after this initial training session and then again after a short time in use. We will also observe the change on the state space that comes about by the MLH utility function ruling out irrelevant state variables.

5.7.1 Initialization Process

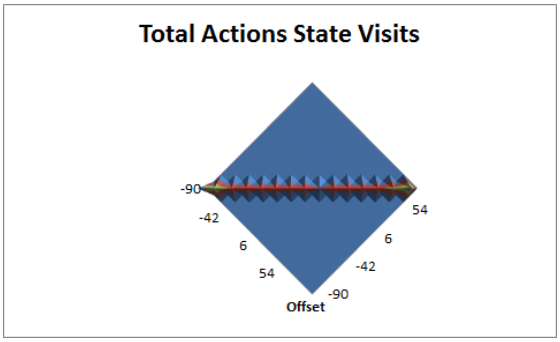

Figure 68 MLH state space exploration during initialization process

Figure 68 shows how much of an MLH state space gets explored during an agent's initialization process. Let us describe this initial exploration process. Initialization actions are performed without other relevant agents in the environment performing any actions. This makes the initial state space exploration a static process i.e. other agents are not performing actions at the same time as this agent. All agents begin in the center of the state space i.e. 0, 0, as the offsets from one agent to all of its neighbors are initially 0. The initializing agent firstly **decrements** its offset to its neighbor agents ten times, performing an offset maintaining action in between each of these. This explores from the center of the state space to the right hand side of the diagram. The agent then performs twenty straight **increment** offset actions, which moves it right over to the left side of the diagram. It then performs a further ten decrement offset actions with an offset maintaining action in between each of these. This returns it to the center of the state space. Thus an important thing to remember about the diagrams that we present in this analysis is that offset decrement actions, which reduce the agent's offset to all of its neighbors simultaneously, moves the agent to the right of the diagram. Offset increment actions move the agent to the left of the diagram.

The initialization procedure is not always necessary during training but it is observed here so that we can give clear representations of the MLH's inner learning process in the diagrams that we present. We can thus far see that a very small percent of the state space has been explored. The reader should note that the size of the state space would be much larger if SAT or the Q-Learning based methods were used instead of Round Robin. This is because their maximum cycle lengths can vary and thus their maximum offsets can be much larger than that of Round Robin.

5.7.2 Agent 3

We now consider intersection agent 3. This agent is downstream to both agent 1 and agent 2. It thus controls the only intersection in the network that is not a border intersection, as only platoons of vehicles arrive on its approaches that have already passed through either agent 1's or agent 2's intersections. What

we will expect to see is that the MLH will consider offsets to both neighbors as being relevant and will thus not rule out either of them. We will first analyze agent 3's MLH immediately after it has been initialized. We then analyze this same MLH after it has been in use for some time so as to see how the learning patterns within it are progressing. We then analyze the MLH's optimization abilities which enable it to remove unnecessary state variables.

5.7.2.1 After Initialization

Figure 69 Intersection agent 3 MLH analysis after initialization

The diagrams in Figure 69 can be best understood by firstly looking at the lower diagram. Each line in the diagram represents the utility value associated with one of the agent's three possible action choices i.e. maintain, increment, or decrement the agent's offset to its neighbors by adjusting its cycle commencement time. As we can see at point offset 0 on the x axis the action with the highest utility value is the increment offset action and thus this is the action that should be selected here for execution. This lower diagram is simply a cross section of the upper diagram. Thus the point 0 on the x axis of the lower diagram actually represents point 0, 0 on the upper diagram. The cross section diagram thus cuts across the upper diagram from point -90, -90 to point 90, 90. In the upper diagram only the actions with the highest utility values are displayed at any point. Thus again we can see that at point 0, 0 the increment offset action should be selected for execution (color coding remains the same between the two diagrams). For both of the above diagrams, as is the case with Figure 68 offset decrement actions move the agent to the right of the diagram and offset increment actions move the agent to the left of the diagram. We can

see from the cross section diagram that when an agent finds itself at point 0, 0 it will increment its offset until it gets to an offset of roughly -30. This can be seen as the point at which the agent can find the highest utility value. At this point it will maintain its offset. If it moves from this position to the left or right it knows which action to perform to get back to the point with the highest utility.

Although the majority of the state space has not been yet been explored (see Figure 68) we can note from Figure 69 that the MLH does contain information regarding the action decisions that should be made in a large portion of the entire state space. This is due to the generalization capabilities of MLH.

Taking into account only immediate rewards as opposed to Q-Learning based utility values would have led to the following action selection diagram.

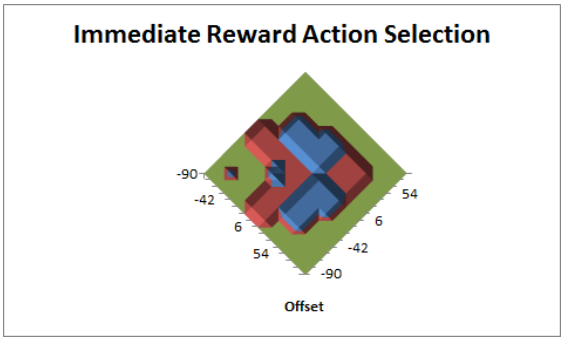

Figure 70 Intersection agent 3 MLH analysis after initialization – immediate reward action selection

This is a very different, and a much less accurate, result from the action selection diagram presented in Figure 69. This highlights to us the importance of using utility values as opposed to simple short term rewards.

5.7.2.2 After Extended Use

We will now look at agent 3's MLH utility function after some extended use. One important thing to note about this is that in the previous sub-section the agent's environment was kept static during the initialization process. The results discussed in this sub-section are of the agent after it has been exposed to the dynamic environment in which all agents are making decisions and performing actions simultaneously. The environment has thus now become a very noisy and dynamic one in which to dwell.

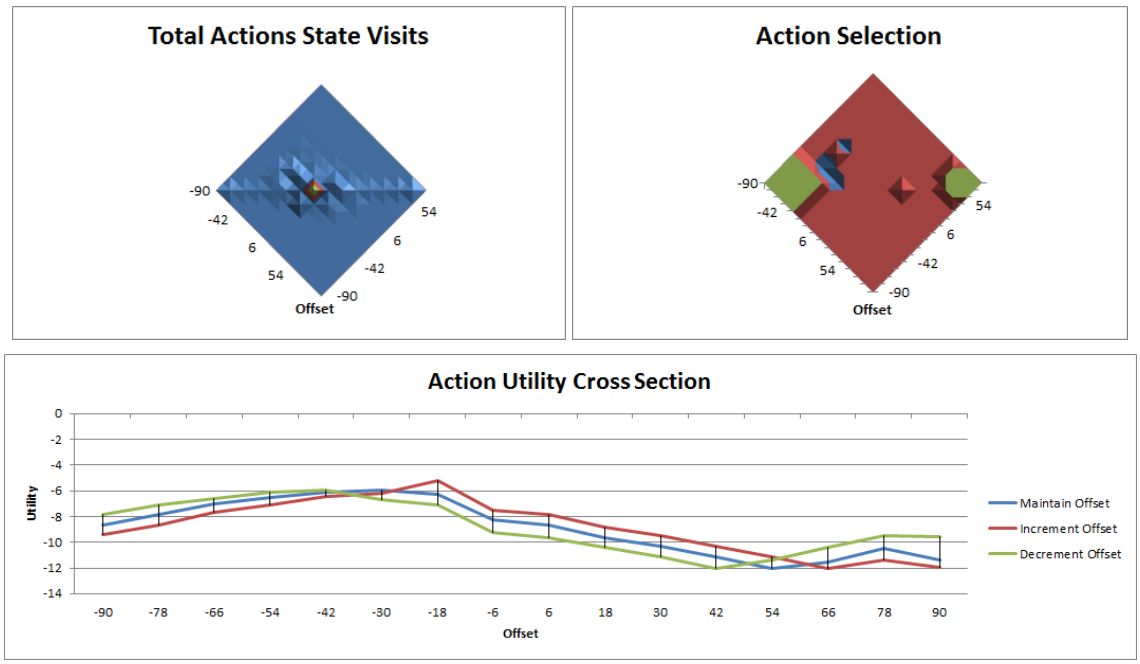

Figure 71 Intersection agent 3 MLH analysis after extended use

From the first diagram we can see that although much more of the state space has been explored the agent dwells particularly in the area of the state space that leads to the higher utility values. From the action selection diagram we can see that the terrain here has also been modified and again reflects that the agent will tend towards roughly the -30,-30 area. Less areas in the state space however now require offset maintenance actions, even though these areas have not been explored. Again this is indicative of MLH's ability to generalize. The action utility cross section diagram again helps us to see from a different angle that the agent will tend towards the -30,-30 area of the state space.

5.7.2.3 Optimized

At this point we would look at agent 3's analysis diagrams after MLH has ruled out any state variables that it deems unnecessary. In the case of intersection agent 3 however it has been deemed to be dependent on its offset to both intersection 1 and intersection 2. Neither of the state variables has thus been abstracted away and thus the diagrams remain as they are. This is in accordance with what we expected to see for agent 3.

5.7.3 Agent 1

We will now consider the MLH utility function of agent 1. Agent 1 is in a very different network location than agent 3. As traffic is flowing from west to east and from south to north we can see that although agent 3 was downstream to both agents 1 and agent 2, agent 1 is only downstream to agent 0.

Agent 1 is however upstream to agent 3. Each of the Qoordination agents act so as to optimize their own local rewards. We would thus expect that agent 1 is dependent on its offset to agent 0 but not dependent on its offset to agent 3 to optimize its reward. In the case of agent 1 we would thus expect to see that its MLH removes from consideration, or abstracts away, the variable stating its offset to agent 3.

5.7.3.1 After Initialization

Agent 1's initial action selection diagram is somewhat different to agent 3's and is given below:

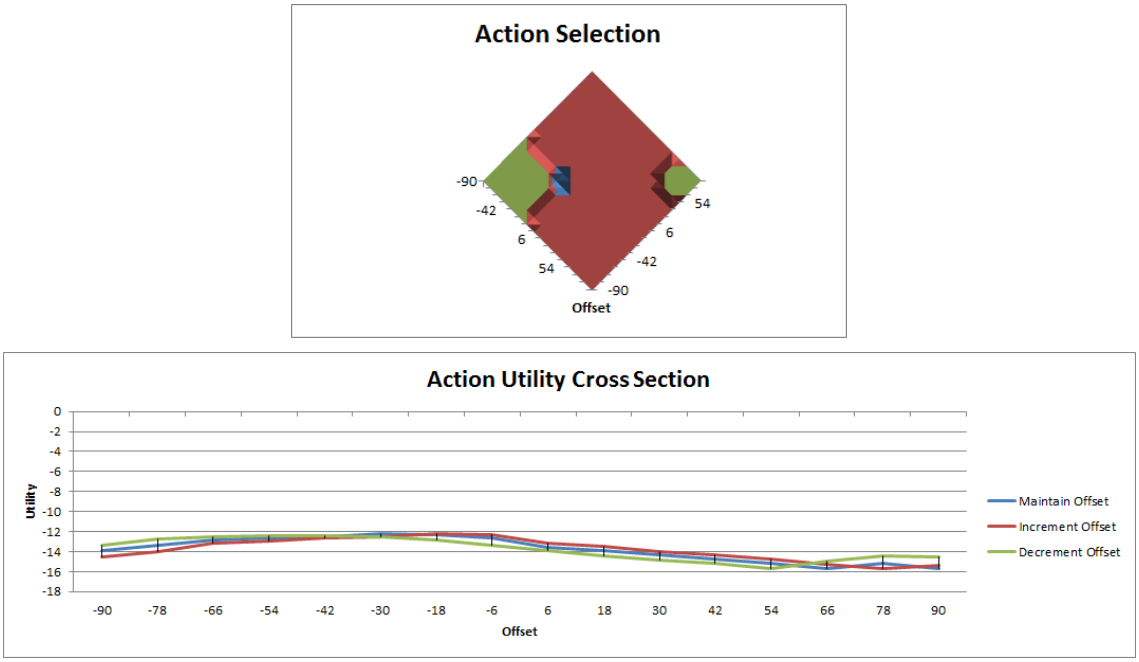

Figure 72 Intersection agent 1 MLH analysis after initialization

We can see here that agent 1 tends towards the -30, -30 area of the state space. This is similar to what we observed for agent 3. The reason for the similarity is that agent 1 is dependent on agent 0 and must maintain this level of offset in order to maximize its own local reward. We know that agent 1 is not dependent on its offset to agent 3, thus this variable does not affect the diagram.

5.7.3.2 After Extended Use

Looking at the MLH after a period of extended use leads to some interesting patterns.

Figure 73 Intersection agent 1 MLH analysis after extended use

We can see from this state space that agent 1 tends to keep an offset of -42 more to one neighbor agent than to another. Thus the agent is starting to be able to tell that it is more important for it to keep an offset of -42 to agent 0 than it is to keep an offset of -42 to agent 3. This can be observed in its updated action selection diagram, where the agent tends to care much more about moving along one axis than the other. We would expect that this would be picked up on by the MLH and taken into account during the optimization process, which will be shown in the next subsection. This will also make the above pattern more clear to the reader.

5.7.3.3 Optimized

The MLH is optimized by abstracting away any state variables that it considers irrelevant to maintaining accurate utility values. The optimized MLH utility function analysis diagrams are shown below:

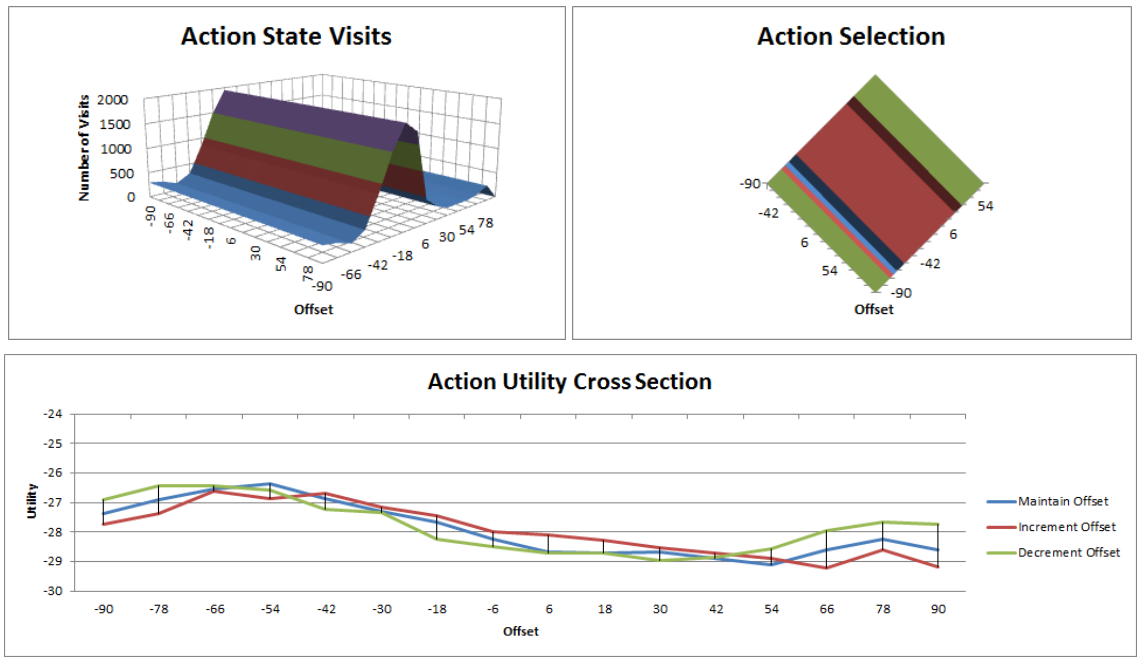

Figure 74 Intersection agent 1 MLH analysis optimized

We can see from these diagrams that one of the offset state variables has been abstracted away and the state space has moved from a two dimensional space to being a one dimensional space. Although the Action Utility Cross Section remains the same throughout this change the other diagrams change significantly. Learning within this single dimension is now done much quicker than in a two dimensional state space.

5.7.4 Agent 2

Agent 2 resides in a network location very similar to that of agent 1. It has two neighboring intersections, namely those controlled by agents 0 and 3. Like agent 1 it is downstream to agent 0 and upstream to agent 3. For this reason it is only dependent on its offset to agent 0 for learning how to maximize its own local reward. We have found that agent 2's MLH analysis graphs are thus very similar to those of agent 1, which have already been presented in the previous sub-section. For this reason we only show agent 2's final optimized action selection diagram as an illustration of this similarity.

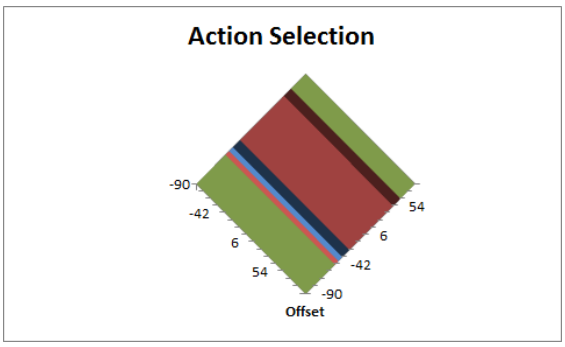

Figure 75 Intersection agent 2 MLH action selection optimized

5.7.5 Agent 0

Agent 0 resides in somewhat of a unique network position in this experiment. It has two neighboring intersections, namely those controlled by agents 1 and 2. Yet it is upstream to both of these intersections and downstream to no intersection. Thus the offsets to both of its neighbors have no direct relevance to agent 0's local reward. It is thus not possible for agent 0 to detect a pattern to learn how to maximize its local reward as no pattern exists. It therefore does not matter to agent 0 which action it chooses to perform at any time. It does however matter to the downstream agents as they are dependent upon their offsets to agent 0. If agent 0 were to constantly execute random actions then agents 1 and 2 would struggle to keep up and maintain coordination. It is for this reason that Qoordination will recognize agent 0 as the key intersection, who is not entitled to perform any offset modification actions. One thing however that we can learn from agent 0's analysis diagrams is that MLH recognizes that there are no dependencies on offset to either neighbors. Both offset variables are thus removed from consideration. This is illustrated in Figure 76 as the state space has become a 0 dimensional space.

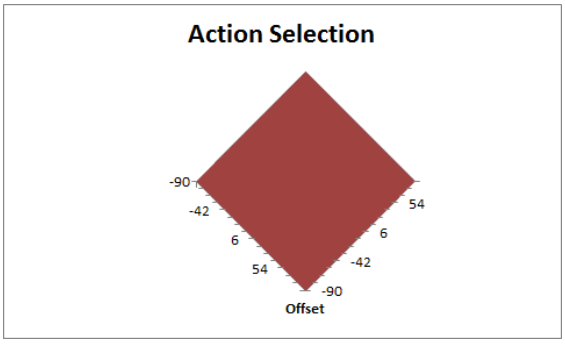

Figure 76 Intersection agent 0 MLH action selection optimized

5.7.6 Conclusion

In this section we have analyzed the learning functionality of the MLH function approximator to ensure that it functions as it was designed to. This analysis was designed so as to meet evaluation objective 3, which in turn addresses design requirement Req3. Through this analysis we have been able to observe that MLH performs generalization such that estimated utility values can be returned even very early in the learning process. These values allow for action selection that leads to high rewards very early in the simulations. We have also seen that MLH performs abstraction very well. In our scenario the different traffic controller agents' MLHs have been able to correctly recognize the offsets to neighboring intersections that are not relevant to them in obtaining a high local reward. These offset variables have then been abstracted out of the MLH so as to increase learning speed without suffering any repercussions with regards to accuracy.

In particular in this section we aimed to evaluate MLH as a method that is suitable for Qoordination. Through the scenario presented in this subsection and our analysis of the results obtained we have achieved this goal. Although MLH is a method of function approximation that has huge potential in being applied to other areas it is not our intention in this thesis to give a full set of tests directly relating to its functioning and performance in comparison to alternative methods such as lookup tables and neural networks. This undertaking is out of the scope of this thesis. We do however see this as a very important endeavor that we fully intend to pursue in future research. The results of this pursuit we plan on publishing in future literature in both the traffic control and machine learning communities.

5.8 Summary

This chapter presented an in depth evaluation of the Qoordination method of intersection controller coordination. After having presented the evaluation objectives and metrics the different traffic control methods to be implemented by the Qoordinated agents were presented. Important parameters set during evaluation experiments were then given. We then described a set of evaluation experiments and discussed and analyzed each of their results in turn. From this analysis of the results we have confirmed that actuated control is not a suitable method for coordination. When intersections using actuated control do attempt to coordinate their action choices so as to establish progressive signal systems their average waiting times increase dramatically, even when optimally coordinated. This is particularly true as the network size increases and as the traffic flow level decreases. Round Robin, SAT, and the Q-Learning based method of traffic control are however suitable methods for coordination. We have found that the positive effects of coordination on traffic flow increase as the network size increases and as the traffic flow levels increase. In networks of 5x5 intersections under heavy traffic flow levels queue lengths were reduced by 63% and average waiting times by 76% when compared to the same traffic control method,

i.e. Round Robin, that did not use any form of coordination. Our analysis has also highlighted that the positive effects of coordination are again increased as the complexity of the main traffic corridor increases. In an experiment that we present with a complex traffic corridor path Qoordination leads to reductions in queue length of 70% and in average waiting times of 83% when compared to uncoordinated control. In this scenario average vehicle speeds throughout the network were increased by 50 times those of vehicles traveling through an uncoordinated network. We have also confirmed through our analysis that with regards to queue lengths and waiting times, which are the metrics that we consider most represent the establishment of progressive signal systems, Qoordination performs similarly to optimal pre-coordinated control. Having analyzed the results of our experiments we then analyzed the MLH utility function for intersections in a 2x2 network. Through this analysis we were able to observe MLH's ability to perform generalization and abstraction so as to decrease learning times.

The purpose of this chapter is to evaluate Qoordination with regards to how well it addresses the two research questions defined in section 1.3. We can see through this evaluation and analysis that Qoordination combined with its unique MLH utility function address these research questions, as well as the research requirements defined in section 3.1 and the evaluation objectives defined in section 5.1. Through this chapter's evaluation and analysis we thus confirm the contributions made by this thesis as stated in section 1.4.

Chapter 6

Conclusion and Future Research

In this chapter we summarize this thesis and review its most significant contributions. We then conclude with a discussion of open research issues regarding this work.

6.1 Summary

Chapter 1 gave a brief description of Reinforcement Learning (RL) and introduced the domain of traffic control. The research questions that this thesis addresses were then introduced, as were the principal contribution of this work.

In Chapter 2 an analysis of the RL and traffic control domains was provided so as to equip the reader with the background information necessary to place the work of this thesis into context. We introduced rational software agents and their environments and then detailed learning algorithms used to optimize agent behavior when the environment is modeled as a Markov Decision process (MDP). After providing background information into the traffic engineering domain we present a number of classical approaches to traffic control as well as Artificial Intelligence (AI) based approaches, with particular focus on RL based approaches.

In Chapter 3 we presented a set of requirements for a novel learning based intersection agent coordination method. We then described the motivations behind the design decisions made during the development of such a method. This method is called Qoordination. We described the design of this method in detail with particular focus on its Multi-Layer Hashing (MLH) utility function.

Chapter 4 presented our simulation-based evaluation platform as well as our Qoordination implementation. The evaluation platform is based upon the industry standard PTV VISSIM microscopic traffic simulator. In developing our Qoordination agents we implement a traffic control framework that offers the flexibility necessary to experiment with different design elements. Being able to experiment with elements such as different learning algorithms e.g. Q-Learning, SARSA, ACRL, ADP, etc. facilitated the design of a suitable final solution. The Qoordination agent implementation interfaces directly with the VISSIM simulator using the programming API that it provides.

In Chapter 6 we presented our evaluation and analyses of Qoordination as an approach to intersection controller agent coordination. This evaluation is performed using our VISSIM simulation-based evaluation platform. After having presented the evaluation objectives and metrics the different traffic control methods to be implemented by the Qoordinated agents were presented. Important parameters set during evaluation experiments were then given. We then described a set of evaluation experiments. We discussed and analyzed each of their results in turn. We found that the positive effects of coordination on traffic flow increase as the network size increases and as the traffic flow levels increase. In networks of 5x5 intersections under heavy traffic flow levels queue lengths were reduced by 63% and average waiting times by 76% when compared to the same traffic control method, i.e. Round Robin, that did not use any form of coordination. These savings are at a similar level to pre-optimized coordination. Our analysis has also highlighted that the positive effects of coordination are again increased as the complexity of the main traffic corridor increases. In an experiment that we present with a complex traffic corridor path Qoordination leads to reductions in queue length of 70% and in average waiting times of 83% when compared to uncoordinated control. In this scenario average vehicle speeds throughout the network were increased by 50 times those of vehicles traveling through an uncoordinated network. We ended this chapter with an analysis of the MLH utility function's ability to perform generalization and abstraction for increased learning rates.

6.2 Contributions

This thesis addresses two main research questions and in so doing makes a number of important research contributions. The first of these questions is: How can autonomous intersection agents learn to coordinate their actions so as to create dynamic progressive signal systems within dynamic transport networks? As a solution to this question we have designed and developed the Qoordination method of intersection agent coordination. Qoordination is a novel Q-Learning based method that enables the creation and maintenance of progressive signal systems along main traffic corridors that run through transport networks. Through analysis of the Qoordination method we have found that it is able to establish progressive signal systems that are able to adapt to changing traffic flow levels and changing

traffic flow directions. We have been able to prove that the positive effects of Qoordination on traffic flow increase as the network size increases, as the traffic flow levels increase, and as the complexity of the main traffic corridors increase. In networks of 5x5 intersections under heavy traffic flow levels queue lengths were reduced by 63% and average waiting times by 76% when compared to the same traffic control method, i.e. Round Robin, that did not use any form of coordination. In an experiment that we present with a complex traffic corridor path Qoordination leads to reductions in queue length of 70% and in average waiting times of 83% when compared to uncoordinated control. In this scenario average vehicle speeds throughout the network were increased by 50 times those of vehicles traveling through an uncoordinated network.

The second research question that this thesis addresses is: How can RL agents rapidly learn accurate utility functions within dynamic multi-agent environments? As a solution to this question we present a MLH function approximation method. MLH is a novel locality-sensitive hashing inspired technique for rapid learning. Its rapid learning abilities stem from its ability to perform generalization and abstraction. We present MLH's design as a method of function approximation that is ideal for use in RL algorithms. After presenting its implementation we analyze MLH's ability to perform generalization and abstraction that enables it to perform rapid learning in large dynamic multi-agent environments such as the transport networks addressed in this thesis.

A further contribution of this thesis is the solid methods used in evaluating the Qoordination method. Evaluation is performed using a novel traffic control evaluation platform. The industry standard VISSIM microscopic simulator is fully integrated into this novel platform so as to ensure accurate and reliable results (Fellendorf & Vortisch, 2010). This platform evaluates the effects of coordination on various traffic control methods in a novel fashion. This is done by automatically generating numerous simulated transport networks of incrementally increasing size. Various methods of traffic control are then run within these networks both with and without Qoordination. These methods of traffic control are as follows: Round Robin (Salkham et al., 2008), a basic SCATS (Sims & Dobinson, 1980) based adaptive approach to traffic control named SAT (Richter, 2006), and a single-agent Q-Learning based traffic control method.

Throughout our evaluation experiments we were also able to observe interesting phenomena that are not as accurately measurable and were thus not included in our results and analysis. An example of one such phenomenon is that the establishment of coordinated traffic control within transport networks leads to the emergence of behavior such as traffic congestion being kept to the edges of the transport network while the center of the network is kept free flowing.

6.3 Future Research

During the Qoordination design process a number of areas that have not yet been addressed and that are not within the scope of this thesis were identified.

The introduction of the Multi-Layer Hashing (MLH) function approximator is a major contributing factor to this thesis. Although Reinforcement Learning (RL) algorithms were initially designed with lookup table based utility functions in mind this approach is limited as it is not able to perform generalization and abstraction. This results in slow learning times. Artificial Neural Networks (ANN) can perform generalization and abstraction and are thus much faster to learn, yet their application to RL is somewhat of an art. Additionally they are a black box approach that is difficult to debug and see what exactly is being learned. In this thesis we developed MLH as a function approximator that is particularly suited to RL. Using MLH RL equations only have to be adjusted slightly. MLH is also able to perform generalization and abstraction. In this thesis we were able to analyze the MLHs of a number of traffic controllers during their learning processes. We observed not only the effects of what MLH was learning but were able to graphically represent these traffic controllers' state spaces with regards to the actions that they would select for execution. We were thus able to evaluate MLH in context of its use in Qoordination. MLH has great potential as a function approximator, and in particular as a RL utility function and as such it deserves a complete evaluation in comparison to alternative function approximators. As future research we intend on performing a full comparative analysis of MLH and related approaches. As well as performing experiments using the simulation based evaluation platform introduced in this thesis we also plan on comparing the performance of MLH based RL with that of the DeepMind deep neural network base RL (Mnih et al., 2015) when applied to general learning. DeepMind have released to the public the deep learner source code as well as the evaluation platform on which it runs. This provides other researchers with the ability to directly compare the performance of their own learning algorithms with that of the deep learner in the same scenarios that were used in their paper published in Nature. This provides a great opportunity to highlight the value of MLH as a RL function approximator in comparison to not only simpler methods like lookup tables but also to a high profile method. This comparison will include such metrics as learning time, processor requirements, memory requirements, ease of setup, etc.

Initial results and analyses that we have had published (Fagan & Meier, 2014) comparing Temporal Difference (TD) to Adaptive Dynamic Programming (ADP) learning algorithms in both single-agent and multi-agent scenarios showed that the rapid learning characteristic of ADP approaches may give them significant advantage over TD approaches in multi-agent settings. These results were obtained using a basic traffic ballancing simulator. To confirm these findings requires an implementation of an ADP based traffic coordination method that can be compared to a similar TD based traffic coordination method. This

design, implementation, evaluation, and comparison is out of the scope of this thesis yet its results could be quite significant as they would show that model based approaches to learning are more suitable to multi-agent learning than model free approaches.

Qoordinated traffic control takes into account grid networks of signalized intersections such as is common in larger cities such as Manhattan. Uncontrolled intersections are not however specifically taken into account and are thus not included within our evaluation platform simulations. The inclusion of uncontrolled intersections into the transport network would overturn the design decision to have the environment modeled as being fully observable. A partially observable Qoordination method would make for a more realistic system that would be more likely to be adapted by traffic engineers. A more diverse set of simulated traffic networks could then be generated for use in evaluation.

Bibliography

Abdulhai, B., Pringle, R., & Karakoulas, G. J. (2003). Reinforcement learning for true adaptive traffic signal control. Journal of Transportation Engineering, 129(3), 278-285.

Barrett, L., & Narayanan, S. (2008). Learning all optimal policies with multiple criteria. Proceedings of the 25th International Conference on Machine Learning, (pp. 41–47).

Bazzan, A. L. (2005). A distributed approach for coordination of traffic signal agents. Autonomous Agents and Multi-Agent Systems, 10(1), 131-164.

Bazzan, A. L. C. (2009). Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. Autonomous Agents and Multi-Agent Systems, 18(3), 342–375.

Bazzan, A. L. C., de Oliveira, D., & da Silva, B. C. (2010). Learning in groups of traffic signals. Engineering Applications of Artificial Intelligence, 23(4), 560–568.

Bohm, C., Railing, K., Kriegel, H. P., & Kroger, P. (2004). Density connected clustering with local subspace preferences. Fourth International Conference on Data Mining, (pp. 27–34).

Brooks, R. A. (1991). Intelligence without representation. Artificial Intelligence, 47(1), 139–159.

Camponogara, E., & Kraus, W. (2003). Distributed learning agents in urban traffic control. Progress in Artificial Intelligence, 2902, 324–335.

Claus, C., & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence, (pp. 746–752).

Cuayáhuitl, H., Renals, S., Lemon, O., & Shimodaira, H. (2006). Learning multi-goal dialogue strategies using reinforcement learning with reduced state-action spaces. The Ninth International Conference on Spoken Language Processing.

Dayan, P. (1992). The convergence of TD (λ) for general λ. Machine Learning, 8(3-4), 341–362.

de Oliveira, D., Ferreira, P. R., Jr, Bazzan, A. L., & Klügl, F. (2004). A swarm-based approach for selection of signal plans in urban scenarios. In Ant Colony Optimization and Swarm Intelligence, (pp. 416–417).

de Oliveira, D., Bazzan, A. L., & Lesser, V. (2005). Using cooperative mediation to coordinate traffic lights: a case study. Proceedings of the Fourth International Joint Conference on

Autonomous Agents and Multiagent Systems, (pp. 463–470).

Dusparic, I. (2010). Multi-policy Optimization in Decentralized Autonomic Systems. Ph.D. thesis, Trinity College Dublin.

Dusparic, I., & Cahill, V. (2010). Multi-policy optimization in self-organizing systems. Self-Organizing Architectures, 6090, 101–126.

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, (pp. 226–231).

Fagan, D., Meier, R., (2015). Dynamic multi-agent reinforcement learning for control optimization, Proceedings of the Fifth International Conference on Intelligent Systems, Modelling and Simulation.

Fellendorf, M., & Vortisch, P. (2010). Microscopic traffic flow simulator VISSIM. Fundamentals of Traffic Simulation, (pp. 63–93).

Forbes, J. (2002). Reinforcement learning for autonomous vehicles. Ph.D. thesis, University of California at Berkely.

Gábor, Z., Kalmár, Z., & Szepesvári, C. (1998). Multi-criteria Reinforcement Learning. Proceedings of the Fifteenth International Conference on Machine Learning, (pp. 197-205).

Gadanho, S. C., & Hallam, J. (2001). Robot learning driven by emotions. Adaptive Behavior, 9(1), 42-64.

Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity search in high dimensions via hashing. Proceedings of the Twenty Fifth International Conference on Very Large Data Bases, (pp. 518–529).

Girianna, M., & Benekohal, R. (2004). Using genetic algorithms to design signal coordination for oversaturated networks. Journal of Intelligent Transportation Systems, 8(2), 117–129.

Guestrin, C., Lagoudakis, M., & Parr, R. (2002). Coordinated reinforcement learning. Proceedings of the Nineteenth International Conference on Machine Learning, (pp. 227-234).

Hiraoka, K., Yoshida, M., & Mishima, T. (2008). Parallel reinforcement learning for weighted multi-criteria model with adaptive margin. Cognitive Neurodynamics, 3(1), 17–24.

Howard, R. (1960). Dynamic Programming and Markov Processes. MIT Press.

Humphrys, M. (1998). Action selection methods using reinforcement learning. Fifth International Conference on Simulation of Adaptive Behavior, (pp. 135–144).

Jouffe, L. (1996). Actor-critic learning based on fuzzy inference system. International Conference on Systems, Man, and Cybernetics, (1, 339–344).

Kailing, K., Kriegel, H.P., & Kröger, P. (2004). Density-connected subspace clustering for high-dimensional data. Proceedings of the Fourth International Conference on Secure Data

Management, 4, 246-256.

Karypis, G., Han, EH., Kumar, V., (1999). Hierarchical clustering using dynamic modeling, Computer, 32(8), 68-75.

Klein, L. A. (2001). Sensor Technologies and Data Requirements for ITS, Artech House.

Kok, J. R., & Vlassis, N. (2006). Collaborative multiagent reinforcement learning by payoff propagation. The Journal of Machine Learning Research, 7, 1789–1828.

Kok, J. R., Jan't Hoen, P., Bakker, B., & Vlassis, N. A. (2005). Utile coordination: learning interdependencies among cooperative agents. Proceedings of the IEEE Symposium on Computational Intelligence and Games, (pp. 29-36).

Kosonen, I. (2003). Multi-agent fuzzy signal control based on real-time simulation. Transportation Research Part C: Emerging Technologies, 11(5), 389–403.

Kriegel, H.P., Kröger, P., & Zimek, A. (2009). Clustering high-dimensional data: a survey on subspace clustering, pattern-based clustering, and correlation clustering. ACM Transactions on Knowledge Discovery from Data, 3(1), 1.

Kuyer, L., Whiteson, S., Bakker, B., & Vlassis, N. (2008). Multiagent reinforcement learning for urban traffic control using coordination graphs. Proceedings of the Nineteenth European Conference on Machine Learning, (pp.656-671).

Lloyd, S. (1982). Least squares quantization in PCM. IEEE Transactions on Information Theory, 28(2), 129–137.

Mirchandani, P., & Head, L. (2001). A real-time traffic signal control system: architecture, algorithms, and analysis. Transportation Research Part C: Emerging Technologies, 9(6), 415–432.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529–533.

Murat, Y. S., & Gedizlioglu, E. (2005). A fuzzy logic multi-phased signal control model for isolated junctions. Transportation Research Part C: Emerging Technologies, 13(1), 19–36.

Natarajan, S., & Tadepalli, P. (2005). Dynamic preferences in multi-criteria reinforcement learning. Proceedings of the Twenty Second International Conference on Machine Learning, (pp. 601–608).

Ormoneit, D., & Sen, Ś. (2002). Kernel-based reinforcement learning. Machine Learning, 49(2-3), 161–178.

Papageorgiou, M., Kiakaki, C., Dinopoulou, V., Kotsialos, A., & Yibing Wang. (2003). Review of road traffic control strategies. Proceedings of the IEEE, 91(12), 2043–2067.

Papavassiliou, V. A., & Russell, S. (1999). Convergence of reinforcement learning with general function approximators. Sixteenth International Conference on Artificial Intelligence, (pp. 748–757)

Peters, J., Vijayakumar, S., & Schaal, S. (2005). Natural actor-critic. Proceedings of the

Sixteenth European Conference on Machine Learning, (pp. 280-291).

Ponsen, M., Taylor, M. E., & Tuyls, K. (2010). Abstraction and generalization in reinforcement learning: a summary and framework. Adaptive and Learning Agents, (pp. 1–32).

Porche, I., & Lafortune, S. (1997). Dynamic traffic control: decentralized and coordinated methods. IEEE Conference on Intelligent Transportation System, (pp. 930–935).

Prashanth, L. A., & Bhatnagar, S. (2011). Reinforcement learning with function approximation for traffic signal control. IEEE Transactions on Intelligent Transportation Systems, 12(2), 412–421.

Prothmann, H., Rochner, F., Tomforde, S., Branke, J., Müller-Schloer, C., & Schmeck, H. (2008). Organic control of traffic lights. Proceedings of the Fifth International Conference on Autonomic and Trusted Computing, (pp. 219-233).

Putha, R., & Quadrifoglio, L. (2010). Using ant colony optimization for solving traffic signal coordination in oversaturated networks. Transportation Research Board Eighty Ninth Annual Meeting.

Richter, S. (2006). Learning road traffic control: towards practical traffic control. Technical Report, Albert-Ludwigs-Universitat Freiburg.

Richter, S., Aberdeen, D., & Yu, J. (2007). Natural actor-critic for road traffic optimisation. Advances in Neural Information Processing Systems. (pp. 1169–1176).

Robertson, D. I., (1969). TRANSYT: a traffic network study tool. Road Research Laboratory Report LR 253.

Roess, R. P., Prassas, E. S., & McShane, W. R. (2011). Traffic Engineering. Prentice Hall.

Rosenblatt, J. K. (2000). Optimal selection of uncertain actions by maximizing expected utility. Autonomous Robots, 9(1), 17–25.

Russell, S. J., & Norvig, P. (2010). Artificial Intelligence. Prentice Hall.

Russell, S., & Zimdars, A. (2003). Q-decomposition for reinforcement learning agents. International Conference on Machine Learning, (pp. 656-663).

Salkham, A., & Cahill, V. (2010). Soilse: a decentralized approach to optimization of fluctuating urban traffic using reinforcement learning. IEEE Thirteenth International Conference on Intelligent Transportation Systems, (pp. 531–538).

Salkham, A., Cunningham, R., Garg, A., & Cahill, V. (2008). A collaborative reinforcement learning approach to urban traffic control optimization. IEEE International Conference on Web Intelligence and Intelligent Agent Technology, (pp. 560–566).

Schrank, D. L., & Lomax, T. J. (2007). The 2007 Urban Mobility Report. Texas Transportation Institute, Texas A&M University.

Shoham, Y., & Leyton-Brown, K. (2008). Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press.

Sims, A. G., & Dobinson, K. W. (1980). The Sydney coordinated adaptive traffic (SCAT) system philosophy and benefits. IEEE Transactions on Vehicular Technology, 29(2), 130–137.

Srinivasan, D., Choy, M. C., & Cheu, R. L. (2006). Neural networks for real-time traffic signal control. IEEE Transactions on Intelligent Transportation Systems, 7(3), 261–272.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine learning, 3(1), 9–44.

Sutton, R. S., & Barto, A. G. (1998). Introduction to Reinforcement Learning. MIT Press.

Taylor, M. E. & Stone, P. (2009). Transfer learning for reinforcement learning domains: a survey. Journal of Machine Learning Research, 10(1), 1633–1685.

Tesauro, G. (1992). Practical Issues in Temporal Difference Learning. Springer.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. Communications of the ACM, 38(3), 58–68.

Thorpe, T. L. (1997). Vehicle traffic light control using SARSA. Masters Thesis, Colorado State University.

Tomforde, S., Prothmann, H., Rochner, F., Branke, J., Hahner, J., Muller-Schloer, C., & Schmeck, H. (2008). Decentralised progressive signal systems for organic traffic control. IEEE Second International Conference on Self-Adaptive and Self-Organizing Systems, (pp. 413–422).

Torralba, A., Fergus, R., & Weiss, Y. (2008). Small codes and large image databases for recognition. IEEE Conference on Computer Vision and Pattern Recognition, (pp. 1–8).

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.

Vlassis, N. (2007). A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. Morgan & Claypool Publishers.

Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine Learning, 8(3-4), 279–292.

Webster, F. V. (1958). Traffic signal settings, road research technical paper no. 39. Road Research Laboratory.

Weiss, G. (2013). Multiagent Systems. MIT Press.

Wiering, M. (2000). Multi-agent reinforcement learning for traffic light control. Proceedings of the Seventeenth International Conference on Machine Learning, (pp. 1151-1158).

Xie, Y. (2007). Development and evaluation of an arterial adaptive traffic signal control system using reinforcement learning. Ph.D. Thesis, Texas A&M University.

Zhang, L., Li, H., & Prevedouros, P. D. (2012). Signal control for oversaturated intersections using fuzzy logic. First International Symposium on Transportation and Development Innovative Best Practices, (pp. 179–184).

Zhang, Y., Xie, Y., & Ye, Z. (2007). Development and Evaluation of a Multi-agent Based Neuro-fuzzy Arterial Traffic Signal Control System. Project Report, Texas Transportation Institute, Texas A&M University.