Decentralized Optimization of Fluctuating Urban Traffic Using Reinforcement Learning

As'ad Salkham

A thesis submitted to the University of Dublin, Trinity College

in fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

June 2010

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.

As'ad Salkham

Dated: June 2010

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

As'ad Salkham

Dated: June 2010

Acknowledgements

I would like to thank my supervisor Prof. Vinny Cahill for his support and guidance throughout the Ph.D. process.

This thesis would not have reached its stage now without the mentorship of Dr. Raymond Cunningham. He has shown a great deal of patience and scientific professionalism in advising and helping me. A big thank you to Ivana Dusparic who has shared the same frustrations of traffic simulations and the jolly feelings when one see agents learning ! I appreciate a lot her witty proofreading many of my chapters and her sense of humour that kept us both from edging insanity.

I would like to thank every member of DSG, old and new ones. The exhaustive list might take a page length and if the affiliated people are added then it is going to be two pages ! So, thank you all for your support and friendship. However, special ones have to be mentioned, Marcin Karpiński, Małgorzata Jaksik, Bartek and Ilona Biskupski, Serena Fritsch and Sanad Ghgaraibeh...many thanks.

There are no words that could show my love and appreciation for my family, their sacrifices for me and their patience. My parents have always stressed on the higher priority for education and they were right. They have raised me with the best they can afford and for that I am deeply grateful. My love and gratitude to my three sisters and cousins for standing beside me for all these years.

As'ad Salkham

University of Dublin, Trinity College

June 2010

Abstract

Increasing traffic congestion levels are causing high worldwide economic, environmental and social costs. Efficient urban traffic control (UTC) is part of the solution to the traffic congestion problem. However, UTC optimization is a challenging task. Urban traffic is characterized by constantly fluctuating traffic patterns. Daily variations in traffic volume and direction, driver behaviour, unexpected emergency situations and traffic accidents all result in traffic fluctuations. Consequently, urban traffic networks exhibit non-stationary behaviour and UTC systems are complex. Furthermore, any local traffic control decisions carried out at a given signalized junction controller may affect both upstream and downstream junctions. Hence, uncoordinated or poor local decisions can negatively impact on the traffic network. Modelling UTC as an optimization problem is also complicated by the heterogeneous interlinked layouts of signalized junctions and the scale of the system.

UTC has been a widely studied problem for a long time. Numerous systems and methodologies have been proposed to address it over the last four decades. Classical UTC systems are either controlled by a dedicated central server or in a distributed manner. The majority rely on complex mathematical and predictive models to optimize specific settings of a given traffic controller. With the increasing costs of congestion, the performance of these systems, which are still in service in the major cities of the world, have prompted questions concerning their effectiveness and adaptiveness in saturated traffic conditions. Other approaches range from rule-based systems and those modelled using fuzzy/heuristic and dynamic optimization techniques, to evolutionary game theory and genetic programming based approaches. However, these approaches are still challenged to provide scalable and yet real-time adaptive and responsive performance. In addition, reinforcement learning (RL) and numerous decentralized RL methods are being increasingly studied for UTC optimization. The nature of RL as an unsupervised learning approach, and particularly Q-Learning, as a model-free learning strategy, allows for incomplex problem modelling and control of the exploration process towards a near optimal solution. Such characteristics are advantageous for developing a real-time adaptive and responsive UTC solution.

The uncertainty present in UTC environments makes the optimization task more challenging. One of the major sources of that uncertainty is the non-stationary nature of traffic. An RL approach to UTC optimization must be designed in a manner through which it is firstly capable of distinguishing between stable situations and secondly able to efficiently optimize for each. Moreover, the performance of existing RL-based UTC approaches is often evaluated using simplified grid-like maps. Some approaches use model-based RL and partially observable markov decision processes (POMDPs) that add unjustifiable complexity. When trying to handle the non-stationary nature of traffic while using RL, strict assumptions are needed, e.g., that a small number of stationary traffic conditions recur, that traffic patterns change infrequently and the independence of such changes from traffic controller decisions. In addition, some of these approaches presume the availability of knowledge that is key to their operation but impractical to obtain from the real world.

Our contribution is a decentralized multi-agent RL UTC strategy that models heterogeneous signalized junctions and optimizes UTC in an adaptive and responsive manner. It is motivated by the lack of a model-free decentralized RL approach for UTC optimization that can deal efficiently with the non-stationary nature of traffic without limiting assumptions and the possibility of taking advantage of the increasing availability of floating vehicle data (FVD). The growing adoption of vehicle-to-vehicle/infrastructure communication and the pervasiveness of different positioning systems both motivate the consideration of FVD as a means of providing a rich view of local traffic conditions. We have designed a UTC optimization scheme based on RL that deploys an adaptive round robin controller agent paired with a non-parametric traffic-pattern change-detection mechanism per signalized junction, namely, a Soilse agent. The Soilse agent optimizes phase timings using RL in a non-collaborative manner. The agent is referred to as SoilseC when it also collaborates with neighbours. It adapts to local traffic conditions and responds to different traffic patterns when required. In order to provide for such responsiveness, it quantifies the degree of change per junction using information about local traffic on incoming lanes and its local performance. Essentially, our design allows for agents to relearn upon detecting a persistent local traffic pattern change. The relearning parameters are mainly based on an average sample of the relevant degree of pattern change. An evaluation of our approach shows its effectiveness against a non-adaptive fixed-time UTC system and a saturation balancing algorithm that emulates the Sydney Coordinated Adaptive Traffic System (SCATS). The evaluation is based on simulations of real Dublin maps of different scale and near-realistic traffic volumes and fluctuations deduced from publications by the National Roads Authority in Ireland.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

This thesis presents a new decentralized approach to online optimization of urban traffic control (UTC) using Reinforcement Learning (RL). In our approach, each RL agent learns to control a specific signalized junction through environmental feedback and potential collaboration with neighbouring agents. Agents adapt to local traffic conditions by learning a sequence of traffic light phases to be used. They respond to fluctuating traffic patterns or unsatisfactory performance by relearning based on a local non-parametric traffic-pattern change-detection mechanism. The novelty of our approach stems from its online decentralized UTC optimization scheme using RL without a priori knowledge of traffic models in an adaptive and responsive manner that deals with fluctuating traffic. Essentially, by providing such an adaptive and responsive UTC scheme we aim to reduce congestion in urban areas.

This chapter introduces RL including centralized and decentralized RL schemes. It also provides a historical background concerning UTC and introduces the relevant facts and challenges in the domain. An emerging source of data for UTC optimization namely, floating vehicle data (FVD) is introduced as well as the common UTC concepts and current trends in UTC optimization. Furthermore, we present our hypothesis which is based on a number of arguments concerning the decentralization of UTC online optimization using RL and on the viability of local non-parametric traffic-pattern change-detection. We also present our contribution that provides a scheme for UTC optimization using RL while dealing with fluctuating urban traffic in a decentralized and online manner. Finally, the organization of the rest of this thesis is presented.

1.1 Reinforcement Learning

The essence of RL can be traced to the manner by which nature's intelligent elements can learn by interacting with the surrounding environment. Sutton & Barto (1998) define RL as "learning how to map situations to actions so as to maximise a numerical reward signal". RL is an unsupervised learning approach in the sense that an agent does not rely on a knowledgeable master that might have specific domain knowledge. On the contrary, agents explore their environment by sensing different situations stimuli and then executing some selected action(s) which result in a feedback in the form of a reward.

Any RL solution is based on two basic elements, namely, a reward function and a value function. Optionally, some RL solutions make use of a model of the environment to predict the reward and next state after taking an action in a given state. The reward function is meant to provide an immediate goodness measure for a certain action in a given state. The value function, as opposed to the reward function, tries to indicate the long-run goodness of a given action, i.e., the expected rewards that can be accumulated over the future starting from the current state. Interaction with the environment eventually provides the RL agent with a policy, i.e., a mapping between all states and their respective best actions at any given time. Moreover, action selection can occur using exploratory strategies, (e.g., ϵ-greedy or Boltzmann (Sutton & Barto, 1998)) or non-exploratory strategies, (e.g., greedy). Finding the limit to which exploration should last is known as the exploration versus exploitation dilemma. Exploitation is the phase during which the agent puts the previously learnt policy into control. The essence of the dilemma is in the fact that an agent cannot run purely on exploration or exploitation otherwise it will be learning forever without actually putting the learnt policy into control or it will allow a given policy to control forever, hence a balance is needed. Q-Learning (Watkins & Dayan, 1992) is a well-established model-free off-policy (explained below) RL strategy based on the concept of discounted expected rewards. An RL agent that uses Q-Learning usually learns with a specific rate $\alpha : 0 \leq \alpha < 1$ and a certain discount rate $\gamma : 0 \leq \gamma < 1$ through a Markov Decision Process (MDP) representation of the environment. It is a model-free approach in the sense that it does not require some a priori likelihood model for the actions that could be executed on the environment. Q-Learning is considered an off-policy strategy as it learns and updates the agent's knowledge even while taking actions that could prove to be non-optimal in the future (Abdulhai et al., 2003). Being an off-policy learning strategy, as well as allowing for short period knowledge updating per action taken, Q-Learning is an ideal candidate for UTC optimization given the non-stationary nature of traffic (Abdulhai et al., 2003).

1.1.1 Decentralized Reinforcement Learning

Classical RL is a centralized optimization approach. This makes problem modelling more difficult as the system's complexity increases due to the increase in the number of system's states that need to be represented which could be also accompanied by an increase in the number of decisions/actions.

The UTC problem, for example, deals with numerous interconnected signalized junctions with some of a heterogeneous road layout. For a relatively small city like Dublin, the city centre has roughly ∼250 signalized junctions that need to be simultaneously controlled. A distributed/decentralized version of RL can be useful (Abdulhai & Pringle, 2003) for such a system while a classical (centralized) RL view poses problem modelling complexity as the network of signalized junctions increases in size. Many decentralized RL approaches where no single RL agent models and controls the global problem have been proposed. They provide optimization approaches of a distributed manner that breaks the global optimization problem into manageable sub-problems. Each RL agent deals with its assigned sub-problem locally with the possibility of collaboration, (i.e., knowledge exchange) with other agents. This could be seen as a specialized Multi-Agent System (MAS) where agents use RL for optimization (Buşoniu et al., 2008; Panait & Luke, 2005). Furthermore, several collaborative RL approaches (Dowling et al., 2006; Kok & Vlassis, 2006; Hoen et al., 2006; Goldman & Zilberstein, 2004; Tesauro, 2003; Guestrin et al., 2002; Ahmadabadi et al., 2001; Abul et al., 2000; Tan, 1998; Hu & Wellman, 1998; Claus & Boutilier, 1997; Littman, 1994) have been proposed and we will discuss them later in Chapter 2. Dowling et al. (2006) they use the term CRL to refer to a specific form of Collaborative Reinforcement Learning. We adopt the term CRL only in describing our framework implementation in Chapter 4, however, our CRL view is different than theirs. We use the term CRL to refer to a scheme where RL agents can collaborate, i.e., exchange knowledge of any nature that can be used in updating the agent's local knowledge besides the use of its local rewards.

1.2 Urban Traffic Control

The history of traffic management arguably extends back to the Roman era. It is interesting to note that the proverb "all roads lead to Rome" is based on the fact that a reference point, in the form of a golden milestone, was positioned in the Forum in the ancient city of Rome. Road builders in Rome, in their turn, used milestones as a form of primitive means to inform road users about their relative location to the golden milestone in Rome (Mueller, 1970). These distributed milestones worked as indicators or signals of reassurance for road users that they were on the right route towards Rome. Although they were static, they were sufficient for road users of that era. Since then, the means to

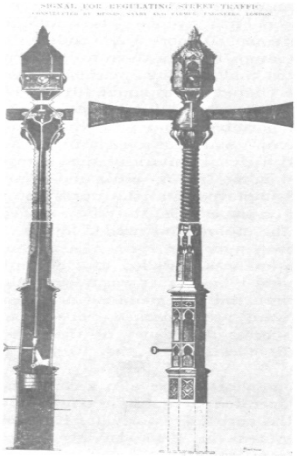


Figure 1.1: Sketch of the world's first traffic signal that was installed on the junction of George and Bridge streets in London in 1868 (Mueller, 1970)

inform and even control traffic created by the increasing number of road users have indeed changed dramatically.

As roads became wider and traffic grew heavier, the need to manage movement within cities and the increasing number of road fatalities became an urgent issue with which to deal. It was in the British parliament in the late eighteenth century that it was first suggested to borrow a method deployed in railways to be used in controlling traffic on roads. A traffic superintendent from the south eastern British railway named J. P. Knight had suggested to Earl Granville that the concept of a railway semaphore signal could be ported onto the road network to allow for traffic control (Ishaque & Noland, 2006). The British parliament agreed to Earl Granville's suggestion and installed the world's first traffic signal (see Figure 1.1) on December 1868 on a junction near the Houses of Parliament in London. That traffic signal was paradoxically put in place to ease road access for members of parliament rather than improve pedestrians' safety. The traffic signal functioned in a way that combined red and green gas lights with semaphore arms. The arms extended horizontally to denote a *stop* signal and on a 45^o angle to denote *caution*. At night, the stop sign was accompanied by a red light on the top while the caution signal was accompanied by a green one. The reader is referred to (Mueller, 1970) for a more in depth history of traffic signals. Henceforth, the terms "traffic signal" and "traffic light" are used interchangeably.

Early traffic signals were controlled by policemen which became increasingly impractical as wider deployment took place in different cities. A greater number of junctions had to be controlled in a manner that was intended to provide better traffic flow within cities. The ultimate goal would

be to provide what is known as a "green wave" or a series of *go* signals along a desired path of controlled junctions. Advances in electronics and computer science have made it possible to devise computerized UTC systems that can manage traffic, in terms of efficient automated operation and performance optimization, on a larger number of controlled junctions, i.e., signalized junctions. Such a system was first deployed in Toronto in 1959 using an IBM 650 computer to control nine signalized junctions (Gazis, 1971). Early UTC systems were centrally controlled and relied on detectors such as magnetic loops, radar and sonar. The main functionalities provided by the control software were electrical actuation of junction controllers, traffic-light state monitoring and detector data processing. The latter data was typically stored for potential offline analysis while some selected data was used for better online control strategies. Such control strategies were mainly based on the concept of synchronizing a line of junctions, usually an arterial road, in order to allow for vehicles to travel at a constant speed with minimal stops. However, those strategies were fixed for certain situations and constrained by the number of controlled junctions. Moreover, with the increasing number of vehicles on roads and the growing scale of urban road networks, the UTC problem has become more challenging. Consequently, the need for more sophisticated and coordinated UTC systems to provide efficient traffic control strategies has arisen. The ultimate goal for such computerized UTC systems is to provide an efficient traffic control strategy that runs in an optimal manner in order to minimize road congestion. This optimality is directly related to achieving minimum vehicle delay, less-interrupted traffic flow or a minimum number of vehicle stops and increased vehicle velocity. Details on the progression of early UTC systems can be found in (Gazis, 1971).

1.2.1 UTC Facts and Challenges

Urban traffic is an evolving problem closely related to population growth and world economic factors. Many countries are seeing an increase in vehicles per capita with each passing year. As far as the Organisation for Economic Co-operation and Development (OECD) countries are concerned, road motor vehicles per thousand inhabitants have increased over the period from 1990 until 2006 in all studied countries except the United States (OECD, 2008) for no clear reason but possibly due to its mature status and increasing environmental public awareness programs. Considerable increases were noticed in countries like Portugal, Iceland, Greece and Poland (see Figure 1.2).

As urbanization is increasing, road networks in different countries are expanding as well. For example, in the European Union (EU), more than 60% of the population are living in urban areas characterized by many more than $10,000$ residents (European Commission, 2007b). The rate at which road networks are expanding varies from one country to another given that some countries already

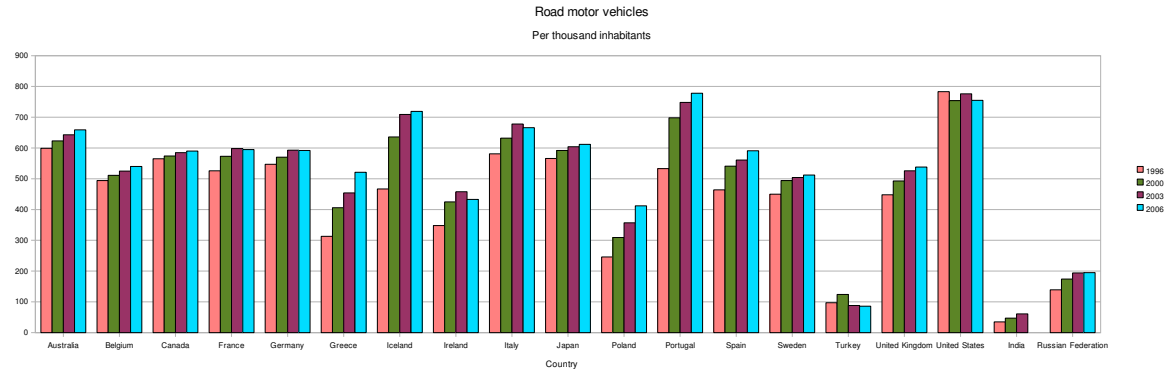


Figure 1.2: Road motor vehicles per thousand inhabitants in selected OECD countries (OECD, 2008)

have mature road networks. The annual growth in road network size can vary from 6% in countries like Korea, Poland, Portugal, Ireland and Greece to a lower rate of 2% in countries with mature road networks like the United States, Germany, Canada, the Russian Federation and the Netherlands (OECD, 2008).

As the network of signalized junctions grew along with the increasing number of vehicles on roads, the problem of providing an efficient UTC system became naturally more complex. Evidently, the problem has not yet been solved, for instance, the United States has roughly 330,000 traffic signals of which 75% can be adjusted to be made more efficient using, but not exclusively, different timing plans (United States DOT, 2007). However, the scale is not the sole issue, road users also exhibit different travelling routines while unexpected emergency and accident situations make traffic networks non-stationary in nature. Such traffic characteristics increase the UTC optimization challenge. Another modelling and control challenge that faces UTC systems is the heterogeneous structure of interlinked signalized junctions. The effects of controller decisions carried out at one junction will propagate in the road network affecting the performance of others, especially, their immediate neighbours. Consequently, the need for a well-designed collaboration scheme is vital in providing efficient UTC systems (Bazzan, 2004).

The negative impact of poor UTC systems is massive and can be essentially summed up in one word "congestion". It is true that better and more efficient UTC systems cannot alone solve this increasing problem but they can surely help to reduce it (European Commission, 2007a; United States DOT, 2007). Congestion causes worldwide environmental, economic and social problems. In the EU alone, congestion annually costs around 1% of the member countries' Gross Domestic Product (GDP) (European Commission, 2007b) and an estimated ∼AU$20.4 billion by 2020 in Australia (Cosgrove

& Gargett, 2007). In 2007, congestion cost the United States ~US$87.2 billion in 439 urban areas calculated based on wasted time and fuel (Schrank & Lomax, 2009). As far as the environment is concerned, congestion is a major cause of air and noise pollution. Urban mobility in the EU contributes 40% of the overall CO_2 emissions caused by road transportation while this percentage increases to 70% of all other pollutants (European Commission, 2007a). These considerable percentages are due to increasing traffic growth and to the stop-go nature of driving in cities despite the advances in vehicle emission reduction technologies (European Commission, 2007b). Furthermore, a recent survey by the Department of Transportation in the United States has shown that 47% of Americans agree that delay caused by traffic congestion is a top community concern (United States FHWA, 2001).

Part of the solution to traffic congestion is evidently better and more efficiently responsive UTC systems (European Commission, 2007b,a; United States DOT, 2007). Adaptive and responsive UTC systems have proved to be promising in many cases in the United States. Compared to previously deployed systems, according to (United States DOT, 2007) for example, a new Texas Light Synchronization program managed to reduce traffic delay by 24.6%, fuel consumption by 9.1% and the number of vehicle stops by 14.2%, all through signal timing optimization and equipment update. In California, a new fuel-efficient traffic signal management program managed to reduce fuel consumption by 8%. Los Angeles' Adaptive Traffic Control System (ATCS) which operates as the city's main traffic control system, managed to diminish average delay by 21.4% and vehicle average number of stops by 31% through real-time response (signal timing adjustment) to traffic demands. The results above encouraged further research in providing more efficient UTC systems in the US through dedicated Federal and State funding programs (United States DOT, 2007). The above advances might have been the result of a long awaited improvement in the poor performance of legacy UTC systems. Recently, new approaches are being considered to come up with "smarter" UTC solutions to deal with the increasing congestion problems, for instance, a recent 2009 governmental report on "Australia's Digital Economy: Future Directions" has identified the use of Artificial Intelligence (AI) and more advanced traffic sensor technologies for developing better UTC systems as a strategic research goal (Commonwealth of Australia, 2009).

The enabling technologies to design and deploy an efficient UTC system that tackles these challenges are increasingly becoming pervasive. The domain that encompasses such technologies is referred to as Intelligent Transportation Systems (ITS) where information processing and communication technologies are being applied to the transportation domain (Yang & Wang, 2007). This ranges from devising better UTC optimization schemes to navigation systems and real-time traffic monitoring. An important driver of ITS applications is floating vehicle/car data (FVD/FCD) and its communication

means. They provide a rich real-time view of traffic status in cities that can be exploited for several applications including UTC optimization.

In summary, it is clear that traffic congestion is a worldwide problem that has been clearly causing economic, environmental and social problems (see statistics above). This problem is worsening with the increase in urbanization, vehicle numbers, population and the possible inefficiency of legacy UTC systems. Different efforts are being exerted to develop more efficient UTC systems that are more adaptive and responsive to traffic changes. However, innovative and "smart" UTC optimization schemes that make use of progressive traffic management technologies like FVD and AI have only recently came into focus.

1.2.2 Floating Vehicle Data

The core idea behind FVD (European Commission, 2003) is to provide different means to communicate various data associated with vehicles in a more pervasive and cost-effective manner using vehicle-to-infrastructure (V2I) or vehicle-to-vehicle (V2V) communication. Such data is usually spatio-temporal, for example, the location of an anonymous (or possibly known) vehicle at a given point of time on the road network. Furthermore, with the increasing availability of in-vehicle sensors, data could range from air pressure levels in tires to fuel consumption and accurate speed data at a given time. Standardization efforts are also playing a major role in helping the spread and adoption of FVD-based technologies and solutions. The International Standards Organization (ISO) and the European Committee for Standardization (CEN) are leading the efforts in providing standards for V2V and V2I communication technologies. Most notably, the Dedicated Short Range Communications (DSRC) (Bai & Krishnan, 2006) and the Continuous Air-interface, Long and Medium Range (CALM) (Williams, 2004) standards that make use of the wireless access in vehicular environments (WAVE) enabling IEEE protocol, namely, IEEE 802.11p (Eichler, 2007). The latter aims at providing a wide platform of different communication technologies working seamlessly together including, for example, DSRC, General Packet Radio Service (GPRS), Global System for Mobile communications (GSM) and International Mobile Telecommunications-2000 (IMT-2000) or 3G.

Traditionally, traffic demand data is gathered through sensors embedded in the road infrastructure such as inductive loop detectors or cameras. With the standardization of FVD technologies and the increasing pervasiveness of wireless positioning systems, e.g., Global Position System (GPS), as well as the considerable investments in V2I and V2V communication technologies; it is now possible to establish a FVD enriched environment with a significantly lower cost compared to the traditional approaches (European Commission, 2003). Moreover, efforts at better positioning systems such as

those of the European Union, have resulted in a promising satellite positioning project namely, Galileo (European Commission, 2001), which is expected to be more accurate than current GPS technology. This will potentially have a positive impact on traffic management solutions (Kuhne, 2003).

Moreover, there has been a recent focus on enriching the set of typical FVD information, e.g., position, speed and time (Messelodi et al., 2009). Through dealing with vehicles as moving sensors, e.g., cameras and traffic level analyzers, typical FVD is enriched with information resulting from vehicle surroundings analysis, e.g., road construction notification and traffic level. The reader is referred to the survey by (Luo & Hubaux, 2004) for more information on FVD.

1.2.3 Common UTC Concepts

There are a number of concepts that are used in describing the functionality within a UTC system. An introduction to some common UTC concepts is provided in this subsection.

- Signalized junction: a junction that is controlled by a traffic light.
- Phase: a phase is characterized by the exclusive set of traffic directions allowed to proceed at a given signalized junction from certain approaches at a given time. Only one phase can be active at a time where all its approaches have a green signal to go.
- Offset (time): the time difference between the start of some phase on a given signalized junction and the start of a different phase on an adjacent signalized junction. Typically relevant when adjacent junctions need to coordinate their phase activation that may affect connecting links.
- Cycle (time): the time needed to complete a sequence of phases on a given signalized junction including offsets.
- Split: the proportioned green time allocated per phase for all phases in a cycle.
- Oversaturation: a situation where links connecting signalized junctions reach their maximum capacity in terms of number of vehicles.

Certain classical UTC systems, as discussed in Chapter 2, base their optimization methodology on tuning signalized junctions timing parameters such as the offset, the cycle time and the split. Some non-classical approaches, however, follow different optimization methodologies based on phase activation decisions and split calculation.

1.2.4 UTC Optimization Trends

Several UTC systems have been proposed over the past four decades. Specifically, two systems, the Sydney Coordinated Adaptive Traffic System (SCATS) (Sims & Dobinson, 1980; Lowrie, 1982) and the Split Cycle Offset Optimisation Technique (SCOOT) (Hunt et al., 1982) have been deployed in many major cities. These systems are based on complex mathematical models to optimize specific timing settings of a traffic controller, namely, the offset, split and cycle time. However, traffic control strategies in such systems are either centrally or hierarchically formulated. Numerous other approaches have been proposed as computational problem solving methodologies have evolved. Such approaches mainly use Dynamic Programming, evolutionary game theory and genetic programming or a combination of those. Others simply use fuzzy/heuristic models and rule-based methods with possible integration with evolutionary approaches. However, RL has emerged as a promising approach for UTC optimization in which true adaptiveness can be achieved (Abdulhai & Pringle, 2003; Abdulhai et al., 2003). We concentrate on decentralized RL that specifically uses Q-Learning for UTC optimization given its scalability and applicability to online (re)learning that allows for the adaptiveness and responsiveness needed by UTC.

1.3 Hypothesis

Our hypothesis is based on the following arguments concerning an efficient UTC system:

- Local traffic signals controlled by RL agents that can adapt and respond to changing traffic are advantageous compared to fixed-time and SCATS-inspired traffic light controllers.
- Designing an RL agent using an adaptive round-robin scheme based on phases to control a given traffic signal is possible.
- Decentralization through assigning a controlling RL agent per signalized junction that collaborates with neighbouring agents can achieve better global performance.
- Detecting traffic changes as they occur is possible based on traffic filtering per lane and the performance of the assigned RL agent without a priori traffic models.
- Responsiveness can be achieved by relearning based on a quantified local degree of traffic change.
- The proposed design does not presume specific sources of sensor information but rather exposes a generic interface.

We evaluate our combined hypothesis using a microscopic simulator that takes as inputs varying traffic patterns simulated on different real maps of Dublin city. The evaluation includes different scenarios characterized by map scale, changing traffic and collaboration. Comparisons are made against scenarios using fixed-time controllers and against a SCATS-inspired algorithm, namely, SAT (Richter, 2006).

1.4 Principal Contribution

This thesis provides a decentralized UTC optimization approach using RL and collaboration schemes, that is efficient, adaptive and yet responsive to the non-stationary nature of urban traffic. Our principal contribution is a scalable scheme in which each signalized junction is controlled by an RL agent that is autonomously capable of detecting unsatisfactory performance and local traffic-pattern change to which it responds by relearning based on the degree of change observed. The RL agent can potentially collaborate with neighbouring agents in order to provide better global performance. With all their characteristics, we name our agents as "Soilse" which means traffic lights in the Irish language. Henceforth, a Soilse agent is RL-based where a SoilseC agent uses RL and collaborates with its neighbours. Furthermore, the approach does not assume any domain knowledge nor predefined models of traffic.

1.5 Thesis Organization

The remaining chapters of this thesis are organized as follows. Chapter 2 presents the state-of-the-art in UTC including classical widely deployed de facto systems, as well as RL and non-RL approaches. The chapter also discusses RL and decentralized RL including the main learning and action selection strategies. In Chapter 3 we detail the design of our UTC optimization agents, namely, Soilse and SoilseC including the pattern change detection mechanism and the relearning strategy. In Chapter 4 we present our implementation using a CRL framework that we built as a C++ library and we describe the interaction between the UTC simulator and the Soilse and SoilseC instances of that framework. Chapter 5 presents our evaluation results based on different axises such as scale, collaboration, responsiveness and action selection strategies. We finally conclude and discuss future work in Chapter 6.

Chapter 2

State of the Art

The thesis merges between significantly wide domains, i.e., reinforcement learning (RL) and urban traffic control (UTC) optimization. This thesis addresses RL-based optimization of UTC, therefore in this chapter we introduce the background necessary for understanding our approach as well as related work to position our contribution and distinguish our approach from existing approaches. In this chapter, we introduce Markov Decision Processes (MDPs) and discuss the essentials of RL and most popular learning and action selection strategies. We also discuss the decentralization of RL. As well, we review different classical, (i.e., currently deployed and de facto) approaches to UTC along with the related work in non-RL-based and RL-based UTC optimization techniques.

2.1 Reinforcement Learning

In this section we introduce RL. We begin by describing MDPs given their close relation to modelling RL problems. We also introduce some well-known approaches to solving MDPs in the sense of seeking an optimal policy.

2.1.1 Markov Decision Processes

Often, RL problems are modelled using MDPs. An agent or any entity that perceives and acts within an environment could cause a new underlying state. Such a state could be the direct result of the agent's actions or due to other factors such as other agents' actions or the natural dynamics of the environment, e.g., the popular prey and predator or multiple predators problem (Kok & Vlassis, 2004). An MDP allows for the modelling of an agent's view of the environment and its interaction with it

through:

- S : a discrete set of states representing the possible environmental settings
- A : a discrete set of actions available to the agent
- $R(s_t, a_t)$: a reward function that returns a reward for taking action a in state s at time t
- $T(s_t, a_t, s_{t+1})$: a transition probability model known a priori that provides the probability $p(s_{t+1}| s_t, a_t)$ of transiting to state s_{t+1} if action a_t is taken from state s_t

Any problem modelled as an MDP must naturally satisfy the Markov property, i.e., the future behaviour depends on the current state s_t but not on the past states. Such a property ensures that a given state captures the effect of a previously taken chain of actions, which allows simpler rules to solve the MDP's optimal policy π^*, where π^* is a mapping from the states to the best actions. It is possible in this case to write one-step formulas that can be, in some form, iterated upon in order to discover π^*. An immediate reward r_{t+1} gives a goodness measure for the action a_t executed in state s_t. It can be calculated based on the reward function $R(s_t, a_t)$ or sometimes using $R(s_t)$ which returns a reward for being in s_t. Such a reward, however, might be insufficient to capture the expected future effect or the long-term usefulness of taking a given action unless it is combined with future rewards. Hence, the concept of future discounted rewards emerges. Naturally an agent will not be likely to wait forever in order to acquire a very high reward, however, it makes sense for it to include future rewards while decreasing their importance as they occur further away in time. Such a behaviour can be achieved using a decreasing discount rate known as $\gamma \in [0...1]$. The Bellman optimality equation (2.1) is a well-known optimality equation based on the concept of discounted rewards and states' expected utility. It is used to find the optimal utility U^* for all states which in many cases is referred to as V^* as well. The Bellman equation aims at optimizing the *value-function* V / U that gives a goodness measure for being in a certain state, or alternatively, the *state-action value-function* $Q(s_t, a_t)$ which provides such a measure per action per state.

$$U^*(s_t) = R(s_t) + \gamma \, max_{a \in A(s_{t+1})} \sum_{s_{t+1}} T(s_t, a_t, s_{t+1}) \, U^*(s_{t+1}) \tag{2.1}$$

Several classical methods have been proposed to solve MDPs. The majority are considered to fall into the Dynamic Programming (DP) paradigm (Sutton & Barto, 1998). Two well-known DP methods for solving MDPs are the value and policy iteration methods. Although the DP paradigm assumes a perfect world model as in MDPs, it is an important basis for understanding RL which does not require such a rigorous assumption.

2.1.1.1 Value Iteration

As the name implies, this method iterates through each state in an MDP using the Bellman optimality equation (2.1) as an update rule in order to reach an optimal policy. The stopping rule for such an iteration is usually based on the maximum difference between subsequent state utility approximations. If that difference is less than $\epsilon(1-\gamma)/\gamma$ then it is guaranteed that the error is less than some value of ϵ. Such an approach relies on an MDP with clearly predefined transition model and state rewards. The method returns the final optimal utility for all states U^*. An algorithm describing the value iteration method is shown in Algorithm (1).

Algorithm 1 The value iteration DP method

V_I(S,A,T,R,γ,ϵ)

$U_t \leftarrow 0$

Do

$\lambda \leftarrow 0$

For $s \in S$

$U_{t+1}(s) \leftarrow R(s_t) + \gamma\, max_a \sum_{s_{t+1}} T(s_t,\, a_t,\, s_{t+1})\, U_t(s_{t+1})$

$\lambda \leftarrow max(|U_{t+1}(s) - U_t(s)|,\, \lambda)$

End

Until $\lambda < (\epsilon\,(1-\gamma)/\gamma)$

Return U^*

The optimal utility for all states U^* can then be used to devise an optimal policy π^* by selecting the action with the maximum expected utility for each state, denoted as $Q^*(s,\, a)$, based on equation (2.2).

$$Q^*(s_t, a_t) = R(s_t) + \gamma \sum_{s_{t+1}} T(s_t, a_t, s_{t+1})\, U^*(s_{t+1}) \tag{2.2}$$

$$\pi^*(s) = argmax_{a \in A(s)} Q^*(s, a) \tag{2.3}$$

The optimal policy π^* can hence be formulated by finding the set of actions of maximum utility for all states, see equation (2.3).

2.1.1.2 Policy Iteration

The policy iteration method consists of two parts through which an agent firstly produces a given policy using the Bellman update equation (2.1) and secondly tries to ameliorate that policy if possible. In essence, the method runs as a sequence of producing policies and testing their stability until an optimal stable policy is found. An algorithm describing the policy iteration method is shown in Algorithm (2).

Algorithm 2 The policy iteration DP method

P_I(S,A,T,R,γ,ϵ)

Initialize U, π

1 **Do**

$\lambda \leftarrow 0$

For $s \in S$

$U_{t+1}(s) \leftarrow R(s) + \gamma \sum T(s_t,\, a_t,\, s_{t+1})\, U_t(s_{t+1})$
$\lambda \leftarrow max(|U_{t+1}(s) - U_t(s)|,\, \lambda)$

End

Until $(\lambda < (\epsilon\,(1-\gamma)/\gamma))$

For $s \in S$

$temp \leftarrow \pi(s)$
$\pi(s) \leftarrow argmax_{a \in A(s)}\, [R(s_t) + \gamma \sum T(s_t,\, a_t,\, s_{t+1})\, U(s_{t+1})]$
If $temp \neq \pi(s)$

Then *goto* 1

End

Return π^*

The value iteration method is a compact version of the policy iteration method. The latter iteratively checks the stability of the resulting policy after a number of value function updates on all states seeking exact convergence. On the other hand, the value iteration method ignores such a rule and acts greedily on the value function updates without seeking exact convergence but still resulting in an optimal policy.

2.1.1.3 Partially Observable MDPs

In certain situations an agent may not be able to determine the state which it is currently in. Such a case is often the result of dealing with an uncertain environment where sensor inputs, fusion and inference techniques are unable to deduce a given state with certainty. Consequently, MDPs can be extended in order to encompass a *belief model* that can provide a probability distribution over the possible set of agent states, namely Partially Observable MDPs (POMDPs) (Kaelbling et al., 1998). For example, an agent can potentially be in three states with a belief state distribution of $< B(s_0) = 0.5, B(s_1) = 0, B(s_2) = 0.5 >$, meaning that the agent can never be in s_1 but has an equal chance of being in either s_0 or s_2 at a given time. As the agent interacts with the uncertain environment, it will naturally need to update its belief model, hence an observation model $O(s, o)$ is used to inform the agent about the probability of an expected observation in a given state. Consequently, the belief model can be determined according to equation (2.4) where α is a normalization factor.

$$\forall_{s_{t+1}} B_{t+1}(s_{t+1}) = \alpha O(s_{t+1}, o) \sum_{s} T(s, a, s_{t+1})\, B(s_t) \tag{2.4}$$

Regardless of the indefinite number of states resulting from the continuous values in the belief model and the intractability of finding an optimal solution in such a case, some approaches have been proposed under assumed constraints in order to provide approximate solutions (Murphy, 1999). .

2.1.2 Reinforcement Learning Structure

Reinforcement Learning (Sutton & Barto, 1998; Kaelbling et al., 1996) is an extensively studied approach to solving a wide range of optimization problems. RL is an unsupervised learning approach that aims at arriving to a setting through which states are optimally mapped to actions, i.e., in a manner that maximizes the long-term expected rewards received after executing a certain action in a given state at a given time. Such a setting achieved by an RL agent constitutes the agent's optimal policy. An RL agent typically discovers its environment through interaction, more specifically, by trial and error. Hence, the learning process through which an RL agent eventually tries to reach an optimal policy, occurs by executing an action in a given environmental state and consequently evaluating the utility associated with that action in that state using the received reward and next state information. Such an RL approach is normally referred to as a model-free approach in the sense that it has no a priori environmental model that specifies the probability distribution on the set of actions allowed in

each state, i.e., a transition model. The contrary is naturally referred to as a model-based approach, which in certain cases predicts/estimates the outcome, in terms of new state and reward. A typical RL agent, see Figure (2.1), represents its local environment through a state-action space in the form of an MDP.

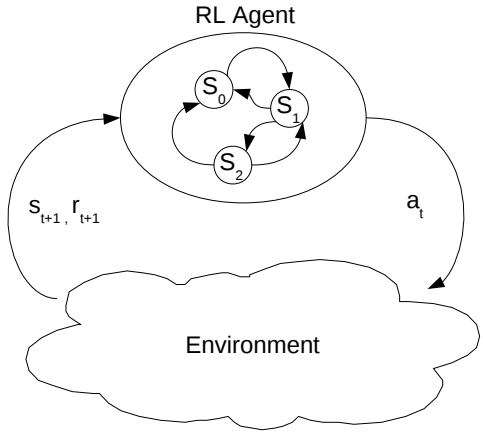


Figure 2.1: A typical RL agent interacting with the underlying environment
s : state, r : reward, a : action

The reward model in RL can be discrete or continuous. One could design a discretized reward model where a constant value is returned based on some goodness conditions. For example, in a grid-world problem, an agent is required to navigate a grid to reach a goal state/square by taking a series of actions from a set of actions $A_{grid} = \{Left, Right, North, South\}$. The agent receives a high positive reward $r_{goal} = 100$ when an action $a \in A_{grid}$ leads to the goal state. On the other hand, any other action a that does not lead to the goal state receives a negative reward $r = -1$. An RL agent trying to find all optimal paths leading to the goal state will try to maximize the expected future rewards in order to achieve its task. Indeed, the agent can receive hints (positive rewards) on the way to its goal square if the problem was modelled in such a way as to hasten achieving an optimal policy. That model will allow for squares positioned one action away from the goal state to return a high positive reward but relatively lower than the goal reward, $r = 50$ per example. Other reward models can be continuous in the sense that they can fall in a given range of values. For instance, an RL-based traffic light controller that tries to arrive to an optimal control policy, which allows for the maximum number of vehicles to pass through, could have a reward model such that $r = number\,of\,vehicles\,passed\,through\,after\,a\,given\,lights\,setting$. In that case, the range of r is relative to the incoming traffic volume. Deciding on whether to use a discrete or a continuous reward model is a domain specific design choice which depends on the nature of the optimization problem.

Essentially an RL agent would model the underlying environment as an MDP. However, in complex dynamic problems such as UTC, it is often very difficult to obtain a definite probabilistic transition model for the MDP to be solved assuming that the resulting state is based on traffic for instance. The same argument applies to obtaining a reward prediction model. However, it is possible to design a reward model that translates environmental feedback. Q-Learning is one of the learning strategies that allows an RL agent, through its value function, to arrive to an optimal policy without the need for a transition model or a reward prediction model.

For an RL agent to function, it relies on a learning strategy, an action selection strategy, a reward model and, vitally, a representation of the underlying environment. We discussed reward models and MDPs as the environmental representation. Onwards we discuss different learning and action selection strategies.

2.1.2.1 Learning Strategies

A learning strategy allows the RL agent to gradually build its knowledge on how to optimally deal with the surrounding environment. That knowledge is cumulatively built through incorporating sensor information in a manner that affects the RL agent's view on the environment. Incorporation is mainly done through a value or a state-action value function update rule of some form.

Q-Learning

Q-Learning was first introduced in the 1989 in Watkins' Ph.D. thesis (Watkins, 1989). Since then, it has been gaining more popularity as a model-free RL technique. Q-Learning falls in the category of off-policy Temporal Difference (TD) learning strategies (Sutton & Barto, 1998). Those strategies are model-free and can update a certain RL agent's policy estimate based on the estimates of other elements in the policy as well as on the incoming rewards. Convergence is assured regardless of the action selection strategy or exploration technique as long as updating all state-action value pairs is continuous. Q-Learning typically behaves in an off-policy manner, which means that it learns even while taking actions that might prove to be non-optimal in the future.

Q-Learning controls the RL agent's learning pace through a learning rate variable $\alpha : (0 \leq \alpha < 1)$ and the level by which it discounts future rewards through a discount rate variable $\gamma : (0 \leq \gamma < 1)$. The Q-Learning update equation is presented in (2.5).

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \, max_a Q(s_{t+1}, a) - Q_t(s_t, a_t) \right] \tag{2.5}$$

$$r_{t+1} : reward\, received\, after\, executing\, a_t$$

A high learning rate implies that the agent is more eager to adopt ongoing changes denoted by r_{t+1} and the future effects of action a_t denoted by $max_a Q(s_{t+1}, a)$ in its updated policy. The RL agent becomes more near-sighted the lower its discount rate is by minimizing the future effect of action a_t denoted by $max_a Q(s_{t+1}, a)$.

Algorithm 3 Generic Q-Learning

Initialize lookup table $\forall Q(s, a)$

QL(S,A,α,γ)

Forall episodes

$s_t \leftarrow s_{initial}$

For each step in the episode **Do**

Select_Execute $a_t : a_t \in A(s_t)$ using some action selection strategy

Receive s_{t+1}, r_{t+t}

$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \left[r_{t+1} + \gamma\, max_a Q(s_{t+1}, a) - Q_t(s_t, a_t)\right]$

$s_t \leftarrow s_{t+1}$

Until $s_t == s_{terminal}$

End

An RL agent using Q-Learning normally keeps a lookup table for all possible combinations of state-action pairs based on the MDP representation of its environment. Its MDP is built without a transition model for action occurrence likelihood nor a reward prediction model. Such a transition model is essentially learnt through interaction with the environment and can be deduced from the state-action values lookup table using some action selection strategy. A generic Q-Learning algorithm is presented in (3). An episode, for example, in a grid-world scenario, could last until the agent arrives to a predefined goal state/square after starting from a different state. The number of learning episodes needed naturally depend on some form of convergence test where the agent terminates if the result of that test is satisfactory. However, in an infinite horizon problem, i.e., a problem that has no specific goal/terminal state, the notion of an episode disappears. In such a case, it is more likely to use a gradually decreasing learning rate paired with a biased action selection strategy that balances between exploration and exploitation, e.g., ϵ-greedy or Boltzmann (see Section (2.1.2.2)).

SARSA

The SARSA RL algorithm gets its name from the knowledge update manner it follows as an on-policy approach. Learning progresses in SARSA from a given state-action pair to another state-action pair and hence the name SARSA, i.e., State-Action Reward State-Action. The learning update rule in SARSA depends on selecting the next action a_{t+1} for the next state s_{t+1} using a common action selection strategy. In contrary, Q-Learning uses the best next action in s_{t+1}. A generic SARSA algorithm is presented in (4).

Algorithm 4 Generic SARSA

Initialize lookup table $\forall Q(s, a)$

SARSA(S,A,α,γ)

Forall episodes

$s_t \leftarrow s_{initial}$

Choose a_t : $a_t \in A(s_t)$ using some action selection strategy

For each step in the episode **Do**

Execute a_t

Receive s_{t+1}, r_{t+t}

Choose a_{t+1} : $a_{t+1} \in A(s_{t+1})$ using some action selection strategy

$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha\,[r_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]$

$s_t \leftarrow s_{t+1}$, $a_t \leftarrow a_{t+1}$

Until $s_t == s_{terminal}$

End

SARSA tends to be a more safe approach as opposed to Q-Learning in the sense that it selects the next action based on a given strategy while Q-Learning risks it by taking actions that might not be optimal but still learns. As a result, Q-Learning is possibly able to reach the optimal policy with less accumulated rewards while SARSA will converge to a near optimal one (Takadama & Fujita, 2005; Sutton & Barto, 1998). Indeed, the design of the reward model is essential in that case.

2.1.2.2 Action Selection Strategies

The RL cycle cannot be complete without affecting the underlying environment through selected actions. For an RL agent to learn by receiving a possibly new environmental state and a reward, it

has to efficiently explore, (i.e., visit different states and try different actions) the space-action space, i.e., the MDP. Naturally, and often in the case of an infinite optimization problem, an RL agent should be able to gradually switch from exploring for an (optimal) policy to exploiting that policy. The role of a biased, (i.e., allows for controlling the exploration period) action selection strategy, (e.g., Boltzmann and ϵ-greedy) hence becomes essential.

Greedy & ϵ-Greedy

The most natural short-sighted strategy that an agent can be following is to always select the action with the maximum positive outcome. Such a strategy is referred to as being greedy given its continuous preference for actions with maximum estimated goodness. However, this type of a strategy alone is problematic for efficient exploration in RL as other possibly less favourable current actions could result in better performance in the long run. To overcome that problem, a randomly greedy action selection strategy was devised, namely, ϵ-greedy. In such a strategy, the current best action is only selected with a probability $(1 - \epsilon)$ where $\epsilon : 0 \leq \epsilon \leq 1$. The greediness of the RL agent is hence defined by the value of ϵ. Moreover, this strategy is independent from the state-action value estimates $Q(s, a)$ in terms of the probability distribution used for selecting the actions.

Boltzmann

The Boltzmann action selection strategy is a customization of the softmax approach (Sutton & Barto, 1998) where the Boltzmann (also known as Gibbs) probability distribution is used to model the action selection strategy. Actions are selected based on a Boltzmann probability distribution built using their $Q(s, a)$ values, see equation (2.6).

$$P(a) = \frac{e^{Q(a)/\tau}}{\sum_{forall\, b \in A}\ e^{Q(b)/\tau}} \tag{2.6}$$

The extent of Boltzmann exploration is controlled by the temperature parameter $\tau : 0 < \tau$. The higher the value of τ is, the more explorative the RL agent is, i.e., actions tend to have nearly equal chances of being selected. As the temperature cools down, the shift towards exploitation becomes greater and the RL agent becomes more greedy. However, deciding on the best initial value of τ is not a straightforward task and could be more of a human intuition.

2.1.3 Decentralized Reinforcement Learning

As optimization problems become more complex in terms of scale, problem modelling in a classical centralized RL manner becomes more difficult and the solution might become intractable. Hence, the need for RL decentralization emerged. Such a decentralization is partially realized through the Multi-Agent RL (MARL) realm. The latter has resulted in a considerable amount of literature.

An important classical differentiation between MARL implementations is presented in (Claus & Boutilier, 1997) between what they refer to as independent learners, where agents learn based on their pure interaction with the environment without realizing the existence of other agents, and joint action learners, where an agent learns a so-called joint action by observing other agents actions and interpreting their local effects. Moreover, an interesting classical study by (Tan, 1998) shows that cooperation among RL agents, if done intelligently, may result in better performance than independent learning. Cooperation there includes communicating agent's local information such as, learning episodes, policies, selected actions, rewards and sensor information. The views presented by (Claus & Boutilier, 1997; Tan, 1998) form the foundations of modern RL decentralization where the single RL agent world has been transformed into a world of RL agents either trying to compete or collaborate.

Concentrating on competitive behaviour, the minimax-Q-Learning algorithm (Littman, 1994), where an agent learns to win as a result of other agent's loss has emerged. An extension to that approach is presented in (Hu & Wellman, 1998). A MARL scheme where coordination among agents is based on having a notion of other agents incorporated in the local state descriptions is presented in (Abul et al., 2000). They concentrate on problems with large state-action spaces where they use generalization and function approximation (Sutton & Barto, 1998). In a coordinated RL scheme (Guestrin et al., 2002), coordination among RL agents is based on coordination graphs where agents select an optimal joint action with one-hop neighbours without searching the large joint action space. A similar Q-Learning specific approach is presented in (Kok & Vlassis, 2006, 2004). Following the idea of learning from the best, a cooperative learning approach for agents using Q-Learning with weighted agent expertness is described in (Ahmadabadi & Asadpour, 2002; Ahmadabadi et al., 2001). Furthermore, a distributed value function learning scheme is presented in (Jeff Schneider, 1999) where an RL agent exchanges its value function estimations with neighbouring agents. An agent in that scheme can learn a value function based on the sum of all other agents' discounted expected rewards. Cooperation through sharing rewards among RL agents is rationalized in (Miyazaki & Kobayashi, 1999) where they provide a minimum precondition to preserve that rationality. On the other hand, an alternative approach to incorporating other agents' rewards or Q-values is presented in (Tesauro, 2003). They present "Hyper-Q" through which an RL agent using Q-Learning can form mixed strate-

gies and predict other agents' strategies through Bayesian inference (Berger, 1993). In a partially observable problem, Goldman & Zilberstein (2004) propose a group of NEXP and P problems where agents share a common decentralized POMDP and try to maximize a global goal through different communication manners.

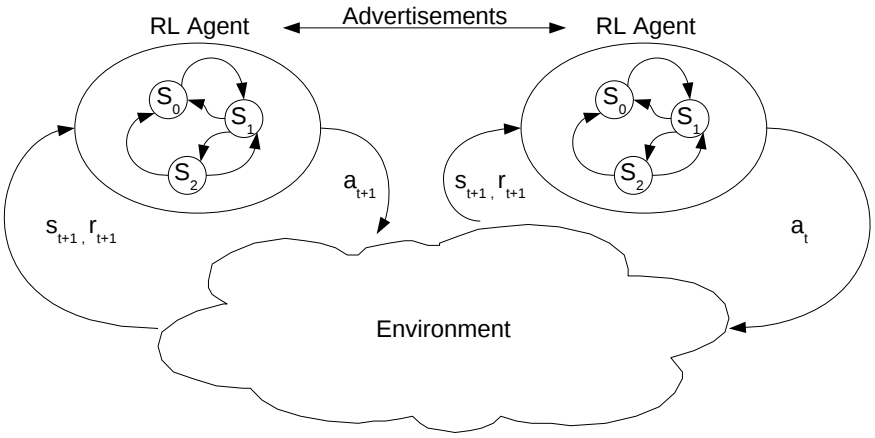


Figure 2.2: A decentralized RL structure

We see the decentralization of RL as a means to break down a global problem into manageable local RL problems. As a result, local RL agents try to act (collaboratively) towards a (near) optimal solution for the common global problem, see Figure (2.2). In (Hoen et al., 2006) a study of the different behaviours (cooperative vs competitive) of learning agents in a multi-agent system (MAS) is provided. As far as the latter study is concerned, we are interested in what they define as *concurrent learning* where each agent learns using a dedicated learning process. Overall, we refer the reader to three surveys (Buşoniu et al., 2008; Yang & Gu, 2005; Panait & Luke, 2005) that could provide a wider view on multi-agent (reinforcement) learning approaches.

2.2 Uncertainty in UTC

The nature of a UTC system is complex and often hard to predict. Numerous factors could shape the unpredictability in a UTC system. Humans' varying behavioural patterns, equipment wearing out and communication noise could all be seen as sources of uncertainty in UTC systems. Consider fluctuating city traffic over different periods of time and the resulting difficulty in adapting control decisions. Such decisions might not only have instantaneous effects but also long-term ones making predictability harder. If we hypothesise the possibility of obtaining 100% accurate communication and traffic information we would still however be unsure about predicting changes in traffic mainly

due to humans, accidents, road works and nature. Interestingly, (Satyanarayanan, 2003) elaborates on uncertainty holistically as in; "it is ironic that in today's all-digital world, uncertainty reappears as a major concern at a higher level of representation". In (Viti, 2006), a thorough study is provided on the uncertainty and dynamics of road users' travelling and delay times. It is argued that uncertainty in a transportation network originates from the variability in supply and demand (Viti, 2006). This could be of a cyclic nature or sporadic, (e.g., hosting world football championship or possibly railway workers strike).

As far as uncertainty is concerned, we are mainly interested in fluctuating urban traffic and the means to generically detect traffic changes online and respond adequately using decentralized RL.

2.2.1 Traffic Patterns

As (Visser & Molenkamp, 2004) put it when discussing the identification of traffic patterns:

"Determining the daily and weekly patterns is a bit of an art, more than a science: results are partly dependent on every individual's own frame of reference (e.g., Is the Thursday before Easter a regular weekday as far as traffic is concerned?)."

Most common approaches to determining traffic patterns are offline approaches that require the analysis of massive historical data and engineering expertise (Venkatanarayana et al., 2007). Furthermore, a question arises concerning the constituents of a traffic pattern. Volume and directionality could be intuitively seen as important characteristics of a given traffic pattern. Several approaches including incident and traffic pattern or state detection have been proposed upon the introduction of Floating Vehicle Data (FVD) technologies (Kerner et al., 2005; Kamran & Haas, 2007; Matschke, 2004; Chen et al., 2007). Most of these approaches centrally process communicated FVD such as travel time and velocity in order to provide a global image of traffic status, or in certain case, traffic accidents (Kamran & Haas, 2007). Also, they mainly rely on road segmentation where specific segments of the road network are individually assigned specific characteristics. In (Matschke, 2004) however, traffic state is estimated using data fusion at the junction level from existing infrastructure sources, (e.g., inductive loop detectors) and signal timings where FVD is only used to help correct such estimations.

Another dimension being explored for general traffic state information gathering is not based on FVD but rather floating phone data (FPD) (Ramm & Schwieger, 2007). Their argument is based on the availability of the GSM infrastructure and on the undesired introduction of additional costs. They rely on matching mobile phones signal strength to signal strength maps provided by GSM network providers. However, we are interested in online traffic pattern change detection that runs locally

without relying on a priori models of traffic.

2.3 Classical UTC Approaches

In this section we present classical UTC approaches. By classical we mean de facto adaptive UTC systems that have been widely deployed over the last four decades in major cities in the world. Miller (1963) was arguably the first to introduce the notion of adaptive traffic control (Bernhard, 2002). The notion of adaptability there was based on basic models to calculate wins and losses from delaying decisions to switch among different traffic light phases. We present two well-known (Klein, 2001) adaptive UTC systems that are still in service in a number of major cities worldwide, namely, SCATS (decentralized) and SCOOT (centralized).

2.3.1 SCATS

SCATS was introduced in the late 1970s after it had been developed by the New South Wales roads and traffic authority in Sydney, Australia (Sims & Dobinson, 1980; Lowrie, 1982). The system was an urgently needed response to the increasing congestion costs in Sydney during that time. Results from its initial deployment showed significant improvement of $35-39\%$ in performance concerning journey time compared to optimized fixed-time signal plans (Sims & Dobinson, 1980). SCATS runs in many major Asian and Australasian cities, such as Sydney, Melbourne, Auckland, Hong Kong, Singapore, Tehran, Doha and Shanghai, as well as, in American cities such as Detroit, Las Vegas, Delaware and Minneapolis. However, it has only been conservatively adopted in Europe, for instance, in Dublin and other cities in Ireland and in Rzeszów in Poland.

SCATS follows a hierarchical hardware control architecture composed of:

- Local controllers: these are microcomputers situated at each signalized junction in order to collect and process data gathered from local sensors, (e.g., loop detectors, usually 5 meters long, or cameras) on every lane per approach. Data processing includes the calculation of headway time, loop occupancy time, space time between vehicles and speed. See Figure (2.3) for a visual explanation of the calculated information. Space time is key to the local controller functionality as it characterizes current traffic flow.

  - They are responsible for the tactical control part in the system. Such control takes decisions to adjust split timings based on analysed data from local sensors but still maintains the same cycle length at each controller.

- Regional masters: each is a computer that controls a network of independent subsystems. A subsystem can be composed of one up to ten local junction controllers. On the software level, the regional computer sees the subsystems grouped into several systems.

  - They are responsible for the strategic control part in system. Such control take decisions to optimize subsystems' different parameters including, cycle time, splits and offsets in order to respond to existing traffic demand.

- Control centre: the supervisor central computer that connects all regional master computers. It allows for overall traffic monitoring including systems, subsystems and local controllers, as well as, data storage and image backups of regional computers. Also, it allows traffic engineers to manually tune or override system settings.

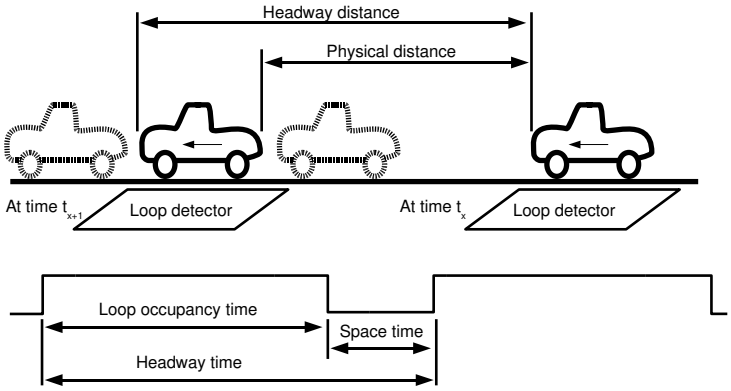


Figure 2.3: SCATS local controller data (Sims & Dobinson, 1980)

Very few algorithmic details are available about SCATS (Head & Sheppard, 1992). However, it is understood that the algorithm mainly relies on the degree of saturation (DS) metric on which it bases adjustment decisions for different cycle, offset and split timings. The DS is based on the efficiency of using the green time at a specific phase. It is calculated by determining the ratio of effectively used green time to the available green time at a given approach. According to (Klein, 2001), "the effectively used green time is the length of green that is just sufficient to pass the platoon of vehicles at the approach had they been travelling at optimum headways under saturation flow conditions." The difference between the effectively used green time and the available one is deduced from the sum of no-load instances on sensors during the green time period compared to that under saturated traffic conditions. Typically, the goal of SCATS is to try to keep the DS to near 90% on the lane with the maximum saturation level. Furthermore, depending on traffic conditions, SCATS allows for

neighbouring subsystems encountering similar or near similar cycle times to unite and form bigger systems or one large system. When the opposite occurs and united subsystems start to encounter different DS levels, they consequently disengage (Head & Sheppard, 1992).

The performance of SCATS is however poor under saturated traffic conditions (Wolshon & Taylor, 1999; United States FHWA, 2008). In a small-scale study in South Lyon city, the overall waiting time, (i.e., delay) in the system was reduced more efficiently under low traffic conditions than under high traffic conditions. Compared to simulated fixed-time control, SCATS average delay per signalized junction was higher under saturated traffic conditions (Klein, 2001).

2.3.2 SCOOT

SCOOT is a centralized UTC system that was developed at the British Transport and Road Research Laboratory (TRRL) (Robertson & Bretherton, 1991). Early prototypes of SCOOT were initially tried in the late seventies in Glasgow and Coventry. Compared against fixed-time plans generated by the Traffic Network Study Tool (TRANSYT) (Robertson, 1969), results have shown an approximate 11% and 16% improvements in terms of delay time under peak and off-peak situations respectively (Klein, 2001). The main difference between TRANSYT and SCOOT is that the first produces optimized fixed-time control plans through offline software simulations while the latter is the hardware realization of a real-time UTC optimization version of TRANSYT (Klein, 2001). SCOOT has been deployed on various scales in numerous cities worldwide. In the UK, it is used for instance, in London, Bristol, Southampton and Edinburgh. It has also been in use in Madrid and in Cyprus. SCOOT has a considerable share of Northern and Southern American deployed UTC systems such as in, Toronto, Santiago and Sao Paulo. Other cities in the world like, Beijing, Dubai, Bahrain, Cape Town and Bangkok also use SCOOT of various scales.

SCOOT uses loop detectors situated upstream from a given junction stop line, normally just downstream from the previous junction. SCOOT bases its performance on three optimization criteria, namely, bandwidth of green waves, average queues and vehicle stops. Green waves are characterized by a series of green signals on a given route where a platoon of vehicles can pass through interconnected junctions without stopping. SCOOT's goal is mainly minimizing the average sum of vehicle queues in a given area and the number of times vehicles need to stop. Consequently, it maintains an on-line model for vehicle queues that is updated periodically in order to determine optimization decisions needed for adjusting split, cycle and offset timings.

SCOOT's optimization routine (Robertson & Bretherton, 1991) is as follows. Shortly before every phase change, the split optimizer decides whether this change should be advanced or postponed

by four seconds or otherwise left to occur without alteration. The offset optimizer is invoked every cycle to evaluate the general performance at a given junction. Similar to the split adjustment, offset optimization decisions dictate four seconds addition or deduction from the current offset or its unalteration. Usually, every five minutes the cycle optimizer decides whether to alter the current cycle time by a few seconds or not. However, SCOOT was reported to degrade in performance under the saturated traffic conditions (Papageorgiou et al., 2003).

2.4 Non-RL UTC Approaches

This section discusses UTC optimization approaches that do not follow the RL scheme. These approaches are grouped according to the design of their control architecture, i.e., centralized, hierarchical and decentralized. The literature available on non-RL UTC is vast, however, we try to cover a sample of representative approaches. An analysis is provided at the end. Furthermore, in Katwijk (2008), a taxonomy is provided of a number of UTC systems based on their architecture, decision making process (e.g., online/offline), traffic prediction model, optimization frequency and their horizon. On the other hand, in (Lin, 1999), UTC systems are discussed based on whether they use offline or online optimization. We discuss non-RL UTC systems grouped by their architecture.

We briefly introduce evolutionary genetic algorithms (Mitchell, 1998) as they are mentioned while discussing some of the systems below. A genetic algorithm is a programming technique that mimics biological evolution as a problem-solving strategy. In general, these algorithms consist of cycles of initializing the population of possible solutions, evaluating each solution according to a fitness function, recombining selected solutions, mutating them and finally evaluating new solutions. This approach enables evolutionary genetic algorithms to seek global optimal performance by tuning their own parameters and adapting to changing circumstances in complex environments.

2.4.1 Centralized

In this section we review centralized UTC approaches, i.e., those where all information processing and generation of timings for signal settings for all traffic light controllers is performed at a single central system point.

2.4.1.1 TUC

One of the main and relatively recent centralized UTC approaches is the traffic-responsive urban control (TUC) strategy (Dinopoulou et al., 2006; Diakaki et al., February 2002; Bielefeldt et al.,

2001). It aims at dealing with saturated traffic conditions in real-time. TUC is based on a store-and-forward model for an urban traffic network that is represented as a directed graph. Essentially, in such a model, vehicles exhibit fixed travel times and are stored at the end of a given link if the incoming traffic is higher than the outgoing traffic. Depending on the signal control decisions, vehicles are forwarded to the next link. TUC's model allows for the use of different programming approaches such as, linear, quadratic and nonlinear programming in order to optimize cycle, offset and split timings per junction. TUC translates the store-and-forward UTC model into a linear-quadratic optimization problem that aims at avoiding traffic spill-back in oversaturated conditions through split tuning. It also tries to maintain a high capacity per junction through cycle time alteration decisions based on a saturation level feedback loop. The offset control decisions in TUC aims ultimately at providing green-waves along arterial roads.

2.4.1.2 DISCO

The Dynamic Intersection Signal Control Optimization (DISCO) (Lo et al., 2001) approach uses a centralized evolutionary genetic algorithm to solve a cell-transmission model (CTM) of the traffic network in an offline manner. The CTM is based on the hydrodynamic theory which models the relationships between density, flow and speed on a macroscopic level.

2.4.1.3 MOTION

MOTION (Busch & Kruse, 2001) is a centralized UTC optimization approach built by Siemens AG, Munich. It optimizes for different signal timings as well as providing incident detection and public transport prioritization. Its optimization algorithm is a multi-step one that firstly gathers traffic volumes and occupancy data, secondly, it models that data on the network level. The third and fourth step are concerned with local junction optimized signal timings and with determining if these timings could serve the global aim of minimizing stops and delays. If found suitable, a slow transition to the new timings is performed. MOTION is proprietary hence details are scarce about the actual optimization algorithm.

2.4.1.4 Others

A centralized UTC approach that models the traffic network using hybrid petri nets (HPNs composed of discrete and continuous PNs) is presented by (Di Febbraro et al., 2004). They propose a control structure through which a supervisor modelled as a HPN coordinates all signalized junctions. Each signalized junction is composed of two controllers, namely, a local and a priority controller. The latter

deals with situations where emergency or public traffic is to be prioritized while the first operates under normal traffic conditions. The local controller aims at minimizing the number of waiting vehicles at a given junction and at equalizing all queue lengths. Moreover, the priority controller at a downstream junction reevaluates its phase timings depending on cost heuristic functions upon being notified by an incoming emergency or a public transport vehicle.

2.4.2 Hierarchical

In this section we review hierarhical UTC approaches, i.e., systems where traffic information processing and decision on traffic signal timings is performed on several hierarhical layers, e.g., on local controllers, regional managers and single central system point.

2.4.2.1 RHODES/COP

The Real-time, Hierarchical, Optimized, Distributed and Effective System (RHODES) is one of the main hierarchical UTC systems that purely relies on DP algorithms (Mirchandani & Head, December 2001). RHODES has a three-level control architecture; network load control, network flow control and, at the bottom, junction control. A parallel three level data architecture feeds each control level with network load predictions, (e.g., capacities, travel times, disruptions), network flow predictions, (e.g, platoon flow) and junction flow predictions, (e.g, vehicle flow). RHODES is able to use data provided by loop detectors, or any form of similar sensors, located upstream from the stop line. It is optionally possible to use data provided by stop-line sensors if available for better queue estimations. The network load controller passes its estimated changes in traffic load to the network flow controller which determines the target signal timings (based on optimization for minimal stops and/or delay) per junction controller. The latter is integral to RHODES functionality and it uses the Controlled Optimization of Phases (COP) (Sen & Head, 1997) model for local junction control optimization. COP is a DP-based algorithm that tries to optimize a sequence of phase timings.

2.4.2.2 UTOPIA

The Urban Traffic OPtimization by Integrated Automation (UTOPIA) (Mauro, 1990) is an Italian UTC system developed at FIAT's research centre in the early 1980s. UTOPIA uses the rolling horizon optimization scheme at the local junction level controller. The local controller keeps a microscopic model of local traffic conditions that allows for an optimization based on different weighted costs for vehicles waiting time, number of stops and queues. Public traffic can also be prioritized using that scheme. On the area level control, groups of signalized junctions are assigned common stage specifics.

2.4.2.3 PRODYN-H

PRODYN-H (Farges, et al., 1983) stands for the French "programmation dynamique" in its hierarchical version. PRODYN-H is composed of two control levels that use an improved forward DP algorithm for minimizing delays based on predicted demand. Limited details are available concerning the hierarchical version but more on its decentralized version is discussed later.

2.4.2.4 Others

Not proposing directly a UTC system but motivated by the increasing number of different traffic "instruments" and especially in the Netherlands, (Katwijk R., October 2002) propose a layered architecture through which traffic instruments are modelled as intelligent agents. These agents coordinate on different levels to form a controller supervision architecture. This architecture comprises a group of network agents on the highest level, followed by route agents and finally a group of measurement agents.

2.4.3 Decentralized

In this section we review decentralized UTC approaches, i.e., systems where traffic information processing and decision on traffic signal timings is distributed on individual traffic controllers and that does not involve centralized elements.

2.4.3.1 PRODYN-D

The decentralized version of PRODYN is referred to as PRODYN-D (Farges, et al., 1983). It uses a rolling horizon optimization approach using an improved forward DP. The horizon is typically segmented into five second sections known as sample time. PRODYN-D makes use of two loop detectors on the upstream and near the stop line in order to gather information concerning vehicles' arrival and queue estimation. This information is used to optimize timings for a given signalized junction for the next seventy-five seconds (horizon duration). The main optimization criterion is minimizing the sum of delays over the horizon. Moreover, neighbouring signalized junctions (usually separated by no more than 200 meters of distance) exhibit coordination by sending control information gathered over the horizon from upstream junctions to downstream junctions. The latter use this information in order to have better vehicle arrival forecasts and hence better optimization of signal timings.

2.4.3.2 ALLONS-D

The Adaptive Limited Lookahead Optimization of Network Signals - Decentralized (ALLONS-D) (Porche & Lafortune, 1997) uses a rolling horizon DP method to optimize for minimum delay per signalized junction. It uses vehicle arrival information gathered from upstream loop detectors in order to choose the suitable phase to be green with predefined limits on maximum and minimum green time for any phase. For instance, a signalized junction that has two phases has a decision space in the form of a binary tree. ALLONS-D assumes the existence of implicit coordination between its controlled junctions given the location of its loop detectors on the upstream and the rolling horizon DP design. However, it provides also a hierarchical architecture version of two levels. The first level is the network level that explicitly communicates certain coordination requirements to the local control. A similar approach to ALLONS-D is the Optimized Policies for Adaptive Control (OPAC) (Gartner, Transp. Res. Record 906, 1983) where the latter uses a different delay calculation model for its optimization scheme.

2.4.3.3 SuRJE

A simulation based approach that uses swarm intelligence for modelling traffic dynamics, i.e., representing vehicles as ants, is presented in (Hoar et al., 2002). This simulation environment is known as SuRJE. In SuRJE, communication among vehicles is done through stigmergy where each vehicle leaves a trace of scents known as pheromones that gradually disappear as time passes. Vehicle speed can hence be determined according to the density of scent traces. Other types of scent traces could denote a decelerating or changing lanes vehicle. Each traffic light in SuRJE uses an evolutionary genetic algorithm to optimize for the minimum cumulative waiting time relative to the current journey time for all cars.

2.4.3.4 Others

An adaptive traffic light scheme that benefits from V2V communication through vehicular ad-hoc networks (VANETs) is presented in (Gradinescu et al., 2007). The adaptive traffic light uses vehicle demand information gathered through communication with vehicles. The local goal is then set to minimize delay by allocating the minimum theoretical optimum cycle time, computed using Webster's equation (Gradinescu et al., 2007). Consequently, the green splits for each phase are calculated in a way that allows for equal saturation levels on all of a junction's approaches.

A reservation-based system for UTC is presented in (Dresner & Stone, 2004). The system assumes each vehicle to be controlled independently by a rule-based driver agent and each traffic light with a

controller agent. Hence, the traffic light controller receives vehicle requests that comprise information concerning vehicle arrival velocity (including minimum and maximum velocity limits), arrival time, direction and vehicle dimensions. Accordingly, the traffic light controller simulates a given vehicle's journey and decides whether to accept or reject the vehicle's request. This decision depends on the availability of slots in the controller's reservation system. The ultimate goal in such a system is to minimize delay. Furthermore, an improved version of the traffic control part is presented in (Dresner & Stone, 2005) and a study for possible multi-agent learning is provided in (Dresner & Stone, 2006).

A distributed game theory-based approach for coordination between traffic light controller agents is presented in (Bazzan, 2004). Controller agents are modelled as "individually-motivated" agents that try to balance between their local interest and the global one. Each agent is assigned a set of predefined strategies with which to play the game. Furthermore, the approach is applied to an arterial road of ten signalized junction agents. A comparison of their distributed coordination approach against a centralized synchronization plan shows better performance in certain scenarios of nearly equal traffic in different directions.

A decentralized logic programming based approach for traffic control is presented in (Felici et al., 2006). This approach uses a logic programming solver, namely, the Leibniz System (Ortega & Planas-Bielsa, 2004) in order to "efficiently" solve the logic problem per signalized junction. A transition graph is used to present the sequence of phases per junction. Transitions are triggered by logical rules that evaluate predicates of congestion levels and phase maximum time per signalized junction. Moreover, in (De Schutter, 1999) a single junction controller is designed to provide near optimal switching scheme by solving a heuristically defined model for the evolution of queue lengths as continuous variables.

2.4.4 Summary

Essentially, centralized approaches to the increasingly complex UTC problem are often of a limited success, especially as providing a scalable responsive UTC behaviour is vital (Bazzan, 2004; Bielli et al., 1994). The general trend is towards the distribution of UTC systems in a hierarchical but increasingly towards a fully decentralized manner. Furthermore, hierarchical UTC systems tend to rely on a DP scheme that uses the rolling horizon technique. Such systems could have limitations when applied on a larger scale given the increasing computational complexity while running in real-time for more than one junction (Cai et al., 2009). As decentralization of UTC is more likely to scale up performance, however, local algorithms need to ensure that better global performance can be achieved, possibly through collaboration. Some of the studied decentralized UTC approaches still rely on the rolling horizon DP which poses questions concerning their efficiency in responding to traffic

changes in real-time given the local microprocessor limitations. Others are inflexible when it comes to the source of information needed for optimization. Some also assume the availability of certain information that might be unrealistic to obtain accurately. Moreover, certain designs of junction controllers using rule-based heuristics and logical programming seem to require human expertise on a junction level. Indeed, such designs have a room for error and their performance compared to the best performance that could be possibly achieved in a given real life scenario is uncertain.

2.5 RL-Based UTC Approaches

This section presents and discusses relevant approaches to UTC that use RL in some form. A number of these approaches use hybrid modelling techniques such as the use of genetic programming or fuzzy neural networks along with RL while others are purely RL-based. In a recent survey (Bazzan, 2009), UTC approaches were classified into three categories; classical (e.g., SCATS), actuated (traffic responsive) and new technologies (e.g., autonomous guided vehicles). Most of the decentralized learning-based UTC approaches where classified as traffic responsive depending on their scale, (i.e., an isolated intersection or more than two intersections) and their support for coordinated optimization. We present several RL-based UTC approaches classified based on the type of technology used.

We briefly introduce fuzzy neural networks (Fullér, 2000) as they are mentioned while discussing some of the systems below. These networks are naturally the result of combining fuzzy systems and neural networks. A fuzzy system is typically a set of parameterized fuzzy rules that are used as an inference engine through interacting with a given knowledge base. On the other hand, a neural network is implemented based on characteristics of biological neurons in order to apply their problem solving techniques to computer learning problems. Neural networks are adaptive, as they learn how to do tasks and create their own connections based on input in a learning phase. Neural networks can be trained using various adaptation and learning algorithms. Hence, a fuzzy neural network is originally a fuzzy system that has been enabled to learn using an algorithm based on neural network theory to determine the parameters of its fuzzy rules by processing a set of observations.

2.5.1 Q-Learning-Based Approaches

In (Abdulhai et al., 2003), whose authors are strong advocates of using Q-Learning for UTC optimization (Abdulhai & Pringle, 2003), results from using Q-Learning for an isolated traffic light controller are shown to outperform a pre-timed scheme by $38-44\%$ for variable traffic flows. Q-Learning either slightly outperformed or was equal to the pre-timed control scheme when traffic flows were uniform

or constant. The reward model used is based on penalization of increasing delay proportional to the queue lengths on all approaches. The junction states are identified through different queue lengths and elapsed phase time. Moreover, an action is characterized by the decision on the length of the next phase time, within practical limits. Noticeably, they do not model the optimization problem as an MDP, instead they use a version of the Cerebellar Model Articulation Controller (CMAC) (Albus, 1975). CMAC is similar to a neural network representation where sensor inputs are mapped to so-called association cells (states) with varying weights. Q-values are used as weights for the state-action pairs and any update on a given pair's Q-value results in an update to the nearby pairs' Q-values. The use of some version of CMAC could be problematic efficiency-wise as junction size and number of states increases. Moreover, no results have been reported about larger scale experiments using multi-agent schemes. A similar approach has been used, in an extended work, for controlling so-called "variable message signs" for the purpose of better ramp metering on a freeway corridor (Jacob & Abdulhai, 2005).

A simple pair of connected traffic light junctions each running a Q-Learning-based agent is presented in (Camponogara & Werner, 2003) where they model and control a small traffic network using a stochastic game scheme. Their results showed that Q-Learning outperformed random and best-effort policies. The reward model is a penalty based on the number of vehicles waiting at a given junction. Moreover, the average number of waiting vehicles was reduced by 30% when both agents were using Q-learning as opposed to it being used by one agent at a time.

Pendrith (2000) proposes a distributed Q-Learning scheme in which an offline optimization aims at controlling vehicle speed. The basic model used is a 3×3 grid of mobile vehicles where the learning agent (vehicle) is positioned in the middle. Vehicles are presumed to be equipped with radar sensors that enable a given vehicle to determine the states of the surrounding vehicles if any. No traffic control strategy was proposed there and the assumption of pervasive radar sensors is quite unrealistic.

More complex RL techniques were used in (Richter et al., 2007). They exploited the Natural Actor-Critic (NAC) (Peters et al., 2005) algorithm that is based on four different RL methods, i.e., policy gradient, value estimation, natural gradient and least-squares temporal difference Q-Learning. In their simplified simulation they had five scenarios and every junction on the grid had four phases. NAC managed to outperform a SCATS inspired technique (namely, SAT) in a 10×10 junction grid simulation while optimizing for vehicle average travel time. However, NAC needed approximately three days of real world time in order to be on par with SAT.

2.5.2 Evolutionary Programing & RL

A combination of evolutionary genetic programing and a Learning Classifier System (LCS) is used in the so-called Organic Traffic Control (OTC) (Prothmann et al., 2008) approach. We consider OTC in this section given the close similarity of LCS to RL. In LCS, a rule-based system composed of classifiers, i.e., a set of (condition, action, value) triplets is used. The system learns by receiving rewards from the environment. A group of classifiers whose conditions match a given environmental stimulus form what is referred to as a "match set". The average values of similar actions are calculated and the action with the maximum average value is executed. Consequently, the reward received from the environment is used in updating the values of matching classifiers comprising the executed action. Furthermore, The OTC architecture per junction is composed of three layers. The top layer uses an evolutionary algorithm in an offline manner that interacts with a given simulator in order to provide new classifiers. The latter could be some genetically enhanced offspring classifiers or those that suit new traffic conditions. The middle layer comprises an LCS that makes traffic control decisions and an observer that feeds the LCS with traffic flows, all in an online manner. The bottom layer is a tunable traffic light controller that can relay traffic sensor data to the upper layer. As far as their evaluation is concerned, they have simulated two signalized junctions of different sizes with a flow of traffic of one peak on three different days. The reference baseline they compare against is a fixed-time controller. Their results show $10-12\%$ improvement in average delay distributed among the three days for the bigger junction while it was $6-8\%$ for the smaller junction. However, on the smaller junction, a significant difference was only noticeable during the peak period. The opposite was true on the bigger junction. Furthermore, it transpires (Rochner et al., 2006) that the OTC architecture top layer uses an offline model-based microscopic simulation. Given the incurred computational complexity and infeasibility of installation per junction, it was suggested to be deployed in a hierarchical manner for a group of junctions. As an extension of the OTC work, coordination among OTC controllers was added (Tomforde et al., 2008). Interestingly, there was no significant improvement concerning average travel time and delay while the number of vehicle stops was reduced. Their simulated experiments were based on an arterial road of five three-phased junctions and on a Manhattan-like grid of six four-phased junctions. Moreover, a common critique of the OTC approach and its coordinated version would be the use of a model-based simulation as a key layer that could affect responsiveness. Such a choice might result in a scalability problem and possible complex coordination schemes. Given their small scale experiments that problem might not have been discovered yet. In addition, it is not clear how the actual learning is happening in the LCS layer. Furthermore, (Cao et al., 1999) also propose a form of RL classifier system to build a distributed learning control scheme for traffic light junctions.

There was no significant improvement in their approach against a random traffic control scheme in a small four-junction scenario.

In order to provide "intelligent" cooperation schemes among RL-based traffic control agents, different forms of RL schemes have been coupled with centrally executed genetic algorithms in several cases (Mikami & Kakazu, 1994) (Yang et al., 2005). The genetic algorithms are used to tune the learning parameters of local controllers in order to provide better global performance where RL is used at the local level. However, the experimental scenarios used were based on a small-scale simulation of four to five junctions.

2.5.3 Fuzzy Neural Networks & RL

A combination of fuzzy neural networks and a form of RL is used to build the hierarchical real-time traffic control architecture presented in (Choy et al., 2003). The architecture is divided into junction controller agents, zone controller agents and regional controller agents. Information flows in a bottom-up manner where junction controller agents pass on traffic state, local signal policy and something referred to as a "cooperative factor". The latter determines the level of collaboration needed based on local traffic conditions. Zone controller agents pass on similar types of information as the lower levels to the regional controller. The zone controller fuzzy-neural model determines the signal policy and the cooperative factor based on an assigned inference engine. These inference engines translate a discretized combination of occupancy, traffic flow, rate of traffic change and local cooperative factors through multiple filtering layers of discretized traffic load and cooperation levels into a final zone signal policy and cooperation factor. All processing layers' outcomes (or neurons' outcomes) are assigned varying weights. An RL module built using a similar fuzzy-neural approach runs in a multistage online manner. This module estimates the state of traffic using some delay estimates and calculates a reward based on current, next and best state. According to the back-propagated reward, neurons alter their output weights using a certain topological update formula, and all agents adapt their learning rates. This work has simulated a traffic network based on a section of Singapore's business district comprising twenty-five controlled junctions. They have reported good results in terms of average stoppage and delay time in two scenarios of single and dual peak(s). Further similar work is presented in (Srinivasan et al., 2006; Srinivasan & Choy, 2006). However, the RL approach they follow is of partial significance and dependant on the nature of the fuzzy neural network representation, which is a complex one in that case. In the typical sense of RL, they do not follow a clear learning nor an action selection strategy, nor do they model RL as an MDP.

2.5.4 Model-Based Vehicle-Centric RL

In a vehicle-centric approach, (Wiering, 2000; Wiering et al., 2004) researched the benefits of using multi-agent model-based RL for traffic control. Their approach is vehicle-centric in the sense that each car estimates its waiting time and communicates it to the nearest traffic light. The traffic-light controllers are RL-based agents that implement a value-iteration DP algorithm. The approach is based on maintaining probability estimates of waiting time per vehicle's destination, and its place at every traffic light controller including the state of that traffic light (green or red). More probability estimates are maintained for the status of each traffic light given a vehicle waiting to go to a particular destination at a given place on the signalized junction. The ultimate goal is to minimize the waiting time for vehicles at all junctions. These probability estimates are used in the traffic light's agent value iteration algorithm to update the value function for expected waiting time. Moreover, they experiment with different local and global communication scenarios where traffic light agents can exchange knowledge for better decision making. In (Steingröver et al., 2005), a very similar approach is presented, where they take into account congestion levels at neighbouring junctions in the local decision making process. It is noticeable, that the last two approaches discussed place some serious assumptions on the type of information that is needed and might not be possible to acquire realistically, especially, if traffic patterns are changing.

2.5.5 Specific RL-Based UTC Approaches for Non-Stationary Environments

We discuss the most significant work in RL that directly addresses the non-stationary nature of traffic from a purely RL perspective. In (Oliveira, et al., 2006), an RL approach to optimizing UTC while responding to traffic volume change and driver behaviour, (i.e., deceleration) is presented. They follow a microscopic simulation approach given that it provides more control of individual driver behaviour which allows the introduction of additional dynamicity in traffic. The driver behaviour model used follows the Nagel-Schreckenberg model (Nagel & Schreckenberg, 1992) that allows for acceleration and probabilistic deceleration (which could hint at overreaction in breaking) on roads broken into cells of five meters length in an urban setting. The main design is divided into two stages, firstly, learning for a given traffic pattern and, secondly, detecting changes in traffic patterns. For learning, they assume that every stationary situation can be defined by a so called "partial model". In such a model, two functions are defined; a transition function that estimates the transition probabilities and a reward function for reward estimation. These two functions are updated for a given partial model

based on the number of times n some action a was carried out in state s. Hence, a truncated n is analogous to the learning rate for the transition and reward functions. Any typical model-based RL can be used to locally optimize for a given partial model, such as Prioritized Sweeping (PS) (Sutton & Barto, 1998). The detection of changes in traffic is based on how well a given partial model can represent the current traffic condition. Hence, an error value is calculated for each partial model that determines its suitability for the current situation. This error value is updated proportionally to n and to the discounted transition and reward functions values for the relevant partial model. Consequently, a group of prediction error estimates are updated for each partial model based on the precalculated error values. An active partial model is declared unsuitable for the current situation if its predicted error estimate becomes higher than a preset threshold. As a result of such a situation, the partial model with the lowest predicted error estimate than the given threshold is activated, otherwise a new partial model is created for that situation.

The experimental setup in (Oliveira, et al., 2006) is based on a 3×3 Manhattan road network of varying link speed limits (54, 36, 18 km/h) where different types of traffic patterns are inserted. An agent is assigned to control each of the nine signalized junctions in the network. Depending on a discretized set of traffic volumes, each agent has nine states showing an empty, regular or full traffic conditions on two incoming approaches. Each agent can select between three predefined signal plans that determines fixed phase timings for traffic travelling specifically from east to west and north to south. They have experimented with scenarios of varying deceleration probabilities; [*zero*, 0.1, 0.2, 0.3] where the number of stopped vehicles throughout the experiment duration was used as a metric. As comparison baselines, they have used fixed-signal plans, greedy controllers, and Q-Learning and PS RL methods. Only in one single scenario, (i.e., where deceleration probability was set to 0.1) their approach outperformed the baselines. In the scenario where the deceleration probability was set to *zero*, their approach performed on a par with most of the baselines. In the two scenarios with deceleration probability 0.2 and 0.3, their approach failed against the greedy baseline. It appears that there is a major issue with their approach's responsiveness to changing traffic situations. They argue that on higher deceleration probabilities, learning agents would only be able to recognize one state and hence the poor performance. If we assume the latter was true, then one would wonder why their approach failed to outperform the baselines under a *zero* deceleration probability (their ideal scenario). Moreover, regardless of the simplistic simulation scale conducted and the use of an advantageous Manhattan road layout, results on how efficient their approach was in successfully detecting traffic pattern changes were not provided and other metrics such as vehicle average waiting and travelling times were not measured.

2.6 Summary

This chapter provided the background required for understanding our approach and related work. It presented RL and its different constituents including popular learning and action selection strategies. Also, the decentralization of RL was discussed. Moreover, we discussed the uncertainty in UTC and identified traffic fluctuations as a main source of uncertainty in UTC, therefore identifying the need to provide a traffic pattern change detection mechanism. We believe, this mechanism should not rely on a priori traffic models and should detect changes in traffic patterns in an online manner.

Two classical UTC systems, i.e., SCATS and SCOOT were presented. Although these systems are deployed in many cities, they have also shown certain limitations in their performance. SCATS performs poorly under saturated traffic conditions (Wolshon & Taylor, 1999; United States FHWA, 2008) despite its adaptive nature, while SCOOT was reported to degrade in performance under the same conditions as well (Papageorgiou et al., 2003).
Various non-RL-based UTC approaches categorized by their architecture, i.e., centralized, hierarchical and decentralized, were discussed. Given the increasing problem modelling complexity, the success of centralized UTC approaches is limited. Several decentralized and hierarchical UTC approaches that use DP were also discussed. These approaches mainly use the rolling horizon DP approach. The latter poses limitations when used in a decentralized manner as local microprocessors might not be able to cope in real-time with fluctuating traffic, especially using the rolling horizon DP (Cai et al., 2009). Moreover, certain approaches using rule-based heuristics and logical programming seem to require high expertise and it is not clear how that can scale.

A number of hybrid modelling techniques that combine RL with, for example, genetic programming or fuzzy neural networks were presented. UTC approaches using Q-Learning have shown promising results in terms of reducing vehicle waiting time. More complex RL-based approach, i.e., NAC, needed approximately three days of real world time in order to be on a par with a SCATS-inspired algorithm, which poses a problem for providing real-time adaptiveness in UTC. Some of the other RL-based approaches assumed the pervasiveness of unrealistic sources of sensor information while others did not use real life maps but rather small-scale simulation of a limited number of junctions. Other model-based UTC approaches that use RL do not take into account the fact that using a priori traffic models given the uncertain behaviour of urban traffic is a strong assumption (Spall, 2003). A common critique for most of the RL-based, whether hybrid or not, approaches is that they do not show significantly better performance that is based on real life maps and do not support online UTC optimization under fluctuating urban traffic conditions. An approach that directly addresses the fluctuating (non-stationary) traffic nature was presented in (Oliveira, et al., 2006) which we discussed

earlier. Mainly, the approach is model-based and it does not show significant performance improvement on the baselines they used even on a simple Manhattan-like road network while also not evaluating for basic metrics such as vehicle waiting time.

After reviewing the related work, it appears that there is a gap/need for model-free decentralized RL approaches for UTC optimization that could respond to traffic fluctuations and adapt to new traffic conditions efficiently. We present the Soilse approach in the following chapter.

Chapter 3

Soilse

This chapter describes the Soilse approach to optimization of urban traffic control (UTC). The Soilse approach models individual traffic light controllers as adaptive RL agents capable of responding to changing traffic patterns that might adversely impact their performance. These agents can make use of collaboration with their neighbours to improve performance. In contrast to previous work on Reinforcement Learning (RL) in UTC (see Section (2.5)), Soilse provides a flexible RL agent design that supports optimization for different traffic patterns using a pattern change detection (PCD) mechanism that causes an agent to relearn based on the degree of pattern change detected. In a non-collaborative setting, each signalized junction is controlled by a dedicated Soilse agent that operates independently. However, in a collaborative setting each signalized junction is controlled by a dedicated agent, which we refer to as a SoilseC agent, that operates in collaboration with neighbouring SoilseC agents. Both Soilse and SoilseC agents make use of a local PCD mechanism to provide for responsiveness in the face of fluctuating traffic patterns. A version of Soilse that did not support pattern change detection was published in (Salkham et al., 2008).

The chapter is organized as follows. First, we describe a set of requirements that should be satisfied by an efficient RL-based UTC optimization scheme. We then provide the motivations for and an overview of the overall design. The PCD mechanism is described including the quantified degree of pattern change (DPC) used as a metric for change. We then describe the signalized junction phases used in the overall design. Finally, the design of the Soilse and SoilseC agents are presented along with the relearning strategy used.

3.1 Requirements

As a consequence to the state-of-the-art discussion, a set of four requirements are identified that serve as guidelines toward the design of our RL-based UTC approach. While some approaches are designed to support a certain requirement they compromise on other essentials. An efficient RL-based UTC system must address certain design requirements simultaneously. However, the objectives used to assess the performance of such a system are identified and evaluated in Chapter (5). The identified design requirements are as follows.

Requirement 1 (Req1): responsiveness; an efficient RL-based UTC system has to be responsive to changing traffic conditions in a reasonable duration. This requires an ability to analyse traffic patterns in real-time while assessing their impact on performance and consequently adapting if needed.

Requirement 2 (Req2): adaptiveness; in order to respond to some detected change in traffic pattern that is adversely affecting overall performance, an efficient RL-based UTC system must be adaptive. This adaptiveness is characterized by the availability of a flexible control model that can be reconfigured to meet the demands of the new traffic situation.

Requirement 3 (Req3): openness, an efficient RL-based UTC system should provide a degree of openness when it comes to the source of traffic sensor information. Depending exclusively on loop detectors or cameras as sensor inputs for decision making is unnecessarily limited. Emerging FVD technologies such as global positioning systems and vehicle-to-vehicle/infrastructure communication may provide a more detailed local view of the traffic situation that could be employed for better RL-based UTC optimization.

Requirement 4 (Req4): collaborative; an efficient RL-based UTC system should support collaboration among signalized junction controllers. Carefully designed collaboration schemes in RL-based UTC systems have shown promising results as opposed to operating without collaboration in terms of providing better system-wide (global) performance (Dusparic & Cahill, 2009a).

The aforementioned requirements will be referred to individually as we address each in the design presented in the rest of this chapter. This includes the design of the PCD mechanism, as well as the design of the Soilse and SoilseC agents.

3.2 Overview and Motivations

The approach to UTC optimization that we follow is an RL-based one. A number of RL-based UTC optimization approaches were discussed in Section (2.5). Originally, the choice of using RL for UTC optimization stems from its support for model-free learning that also provides off-policy learning strategies like Q-Learning. The latter has been proved to be beneficial for the optimization of UTC (Abdulhai et al., 2003). Hence, an RL-based approach should learn a near-optimal mapping of states to actions, as opposed to having an a priori model for that mapping, through feedback and interaction with the environment. This is advantageous in such a complex problem as UTC given the non-stationary behaviour of traffic. This aligns with what Spall (2003) argues in relation to providing a traffic control scheme based on models of traffic flow: "which, given the highly nonlinear and uncertain aspects of human behaviour, is a virtually hopeless task in complex multiple-intersection networks."

An approach to RL-based optimization of UTC in a non-stationary environment is provided in (Oliveira et al., 2006) and was discussed in Section (2.5.5). The approach that they provide tries to optimize for a given stationary traffic situation using a model-based RL approach based on what they refer to as "partial models". The latter are characterized by a transition-probability and reward-estimation function for a given stationary traffic situation. The manner (referred to as change detection) through which they try to handle the non-stationary behaviour of traffic is by switching among learnt partial models depending on a suitability test that is based on their model error calculation. However, even in small-scale simulations on a simple Manhattan road network, it is not clear that their approach outperformed the baselines used. Essentially, the approach that they provide is a model-based one with a change detection mechanism interleaved with what they refer to as partial models. In addition, their approach responds to changes in traffic in an abrupt manner that replaces a given partial model by another. In contrast, the approach that we follow is a model-free one that minimizes functional interdependency between local UTC optimization and traffic PCD. In addition, our approach responds to detected changes in a continuous learning manner that does not require storing or switching between learnt models.

In order to provide an efficient RL-based UTC system that addresses all of the requirements presented in Section (3.1) we divided the signalized junction control problem into two main constituents, an RL-based UTC optimization agent and a separate PCD mechanism, see Figure (3.1). In the rest of this section an RL-based UTC optimization agent is referred to as an agent.

Both constituents combined satisfy all the requirements presented in Section (3.1). Particularly, a PCD mechanism contributes to the satisfaction of the responsiveness requirement ($Req1$) by providing the means of detecting local traffic changes that adversely affect the performance of the controller agent

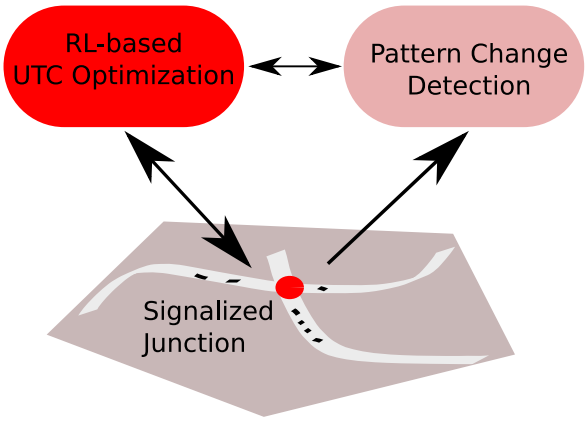


Figure 3.1: Design overview

at some signalized junction. The PCD should quantify the change as it occurs in a way that allows the agent to respond adequately by adapting to the new traffic situation and hence satisfy *Req*2. The PCD mechanism should operate using various sources of sensor information that describes the underlying traffic situation as specified in *Req*3. It also should not rely on a priori models of traffic as this may limit its online/real-time responsiveness. Given these characteristics, an online nonparametric change detection mechanism that quantifies the degree of traffic pattern change affecting the agent's performance can be used. Such a mechanism is nonparametric in the sense that it does not rely on a specific distribution (distribution-less) for incoming traffic on a given signalized junction. Various nonparametric statistical tests were assessed in this case, e.g., Kolmogorov-Smirnov and Cramér-von Mises (Schumacher, 1984). These tests work in way that compares two given samples of data and produce a value, namely, $0 \leq P\_value \leq 1$ that determines the acceptance or rejection of the null hypothesis when compared against some significance level. The null hypothesis typically suggests that the two samples are drawn from the same distribution. However, after assessing the aforementioned nonparametric statistical tests, it appeared that they were not suitable for the online representation of local traffic pattern changes. They have also shown to be sensitive locally especially given the nature of urban traffic data. Consequently, we refocussed our attention towards an analogous (to some extent) area to urban traffic that is computer network traffic and especially anomaly detection (attacks or intrusions of a certain type) (Thottan & Ji, 2003) in computer networks. This suggested a technique for PCD that is based on sequential analysis of data series, namely, cumulative sum of squares (CUSUM) (Oh et al., 2005) that can identify change points in data variance (σ^2). CUSUM is a nonparametric change detection technique that has been shown to be accurate in helping to detect

flooding attacks in a short duration (Thottan & Ji, 2003; Wang et al., 2002; Siris & Papagalou, 2006). Flooding attacks represent a change in the network traffic pattern where, for example, an unusual number of requests can be directed to a given server. This is seen as analogous to change in traffic on road networks. We hence adopted CUSUM in developing our PCD.

The agent has to satisfy all of the requirements specified in Section (3.1) in conjunction with PCD. Especially, in order to provide for the adaptiveness specified in *Req*2, the agent, being RL-based, must be designed in a way that allows it to adapt to a new traffic situation. This normally happens after the PCD initiates a need to respond when some traffic change that is adversely affecting the agent's performance is detected. A natural fair design we follow is based on an adaptive round-robin that learns a sequence of phases of different durations suitable for the new traffic situation. The adaptiveness is enhanced by allowing agents to learn to skip unnecessary phases which saves the wasted time as opposed to SCATS, for instance, that has to give all phases a certain durations. Also, by avoiding free phase selection from a group of phases in no particular order, we minimize the risk of starvation.

A crucial part needed for adaptiveness is the ability of the agent to relearn and hence adopt a different control policy given a new traffic situation. As relearning is naturally expensive, controlling the extent and duration of the agent's relearning is important. Given that the agent is RL-based, this can be done through its learning and action selection strategies, e.g., Q-Learning and ϵ-greedy respectively, whose parameters can then be controlled in a manner that specifies the relearning/exploration duration and their initial exploration related parameter values, e.g., learning rate α and ϵ of the ϵ-greedy. In addition, PCD provides a degree that quantifies the traffic pattern change along with the agent's performance. The agent can then use this degree to calculate new exploration-related parameters, relevant to the learning and action selection strategies in use, in order to initiate relearning. Being responsive and adaptive on the local level is important however, in order to satisfy *Req*4, collaboration is needed. Hence, the agent supports exchanging knowledge with neighbours and allows for its local incorporation. Both PCD and the agent expose generic interfaces for receiving sensor information about traffic hence satisfying openness needed by *Req*3. Regardless of the manner sensor information is gathered or actuation is done, these interfaces guarantee that the agent will continue to function as long as they are implemented.

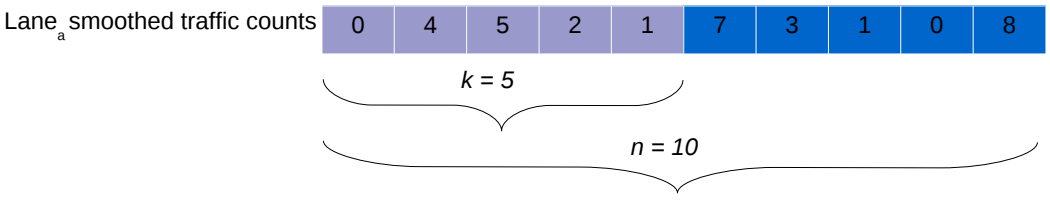


Figure 3.2: Example - CUSUM samples

3.3 Pattern Change Detection

This section presents PCD mechanism that helps satisfy *Req*1 for an efficient RL-based UTC. The motivations behind the design choices were discussed in the previous section. The proposed PCD mechanism is used to detect variance (σ^2) change points on each lanes' incoming traffic (data) using CUSUM and also incorporates knowledge of the agent's performance with that change at each signalized junction. We choose junction lanes as the level of granularity at which to identify changes since lanes represent not only the traffic load but also its presumed directionality. Therefore, this approach captures not only changes in traffic load but also changes in traffic directionality. The performance of an RL agent controller can be naturally assessed based on its recent rewards as they represent the goodness of its local behaviour. Hence, we use recent rewards history to provide a metric for the overall near past agent performance.

3.3.1 Design

The design is based on multistage lane-centric filtering, see Figure (3.3). The incoming traffic count per lane on a given junction is sampled and filtered using a moving average filter in order to produce a smoother input (since we assume a fine-grained initial input as low as a reading per second over a minute time) for the second stage. The output is then passed to the CUSUM (Oh et al., 2005) filter that identifies changes in traffic variance (σ^2) on a given lane. CUSUM is a well-known sequential analysis technique that can indicate change points in data variance, see Equation (3.1). CUSUM is advantageous in this case as it is a nonparametric sequential change detection technique that does not require a predefined traffic model.

$$CUSUM_{k,n}(Lane_a) = \left| \frac{\sum_{i=1}^{k} X_{a\,i}^2}{\sum_{j=1}^{n} X_{a\,j}^2} - \frac{k}{n} \right|;\ for\, 1 \leq k < n \tag{3.1}$$

The time series $T_{counts} = \{X_{a\,1}, X_{a\,2}, ..., X_{a\,n}\}$ is formed from the outputs of the first filtering stage, i.e., a series of size n of smoothed traffic counts for lane a. The value of k represents the lagging sample size, see Figure (3.2). Essentially, CUSUM works by comparing the sum of squares of a portion

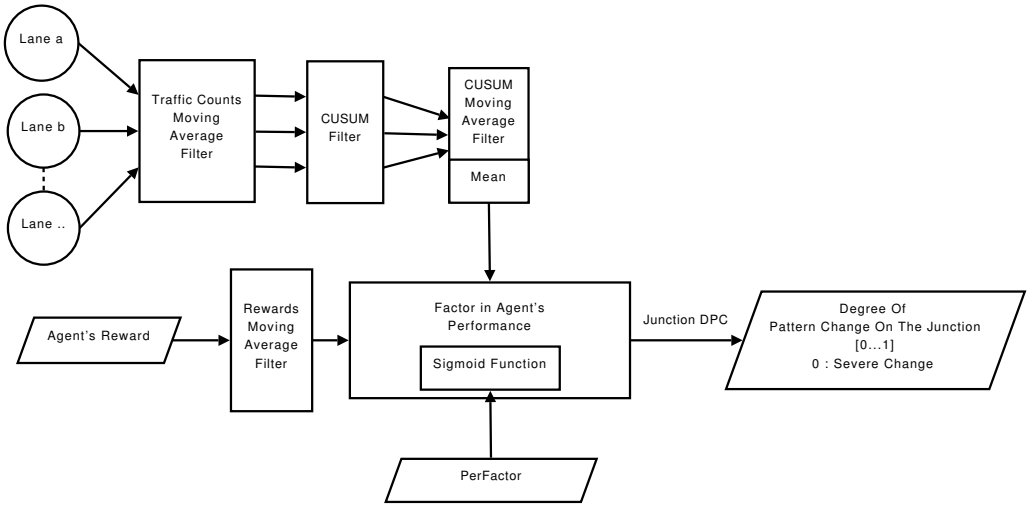


Figure 3.3: Junction PCD high-level scheme

of a given sample against the sum of squares of the whole sample. This allows it to figure out what is the proportional relevance of the smaller sample of size k on the whole sample of size n. This technique has proven to be accurate in identifying change points in data variance (Oh et al., 2005). Another moving average filter is applied on the CUSUM filter outputs and the mean of these is then calculated as the final representation of the degree of pattern change (DPC) in incoming traffic for a given junction.

The DPC at this level does not include a notion of how the possible change is affecting the controller agent performance at a given junction. Therefore, as we are interested in changes that affect the agent's performance, we need to incorporate the controller agent's performance in the DPC, which can be then used in the reparameterization process of the agent for relearning. Consequently, this will allow for agent responsiveness (see $Req1$). A natural metric for the agent performance is the moving average of rewards (MAR) over a given time window. As rewards are an intrinsic indicator of RL agents performance in the first place, we benefit from their availability without introducing an extra artificial metric of performance. In order to produce a final combined change degree that incorporates the agent performance as well as the pattern change degree, the product $(MAR \times DPC)$ is used. Furthermore, in order to confine (squash) that degree to a known range we apply a sigmoid function, see Equation (3.2), that has a known range of $[-1, 1]$.

$$DPC_{squashed} = tanh((MAR \times DPC)/\, PerFactor) \tag{3.2}$$

Parameter	Description
CUSUM: n	The size of the full CUSUM sample of traffic counts per lane.
CUSUM: k	The size of the sub-sample whose sum of squares is compared against that of the complete sample of size n.
Traffic counts sample size	First layer of sensitivity control in the PCD on the raw data, i.e., the moving average of traffic counts per lane.
CUSUM output sample size	Second level of sensitivity control in the PCD, i.e., on the CUSUM output per lane.
MAR sample size	Determines the PCD's sensitivity to the agent's performance.
$PerFactor$	Scales down the input data for the DPC's sigmoid function. Relative to the range of $(MAR \times DPC)$.
JCT	Holds the threshold value used in detecting changes in the traffic pattern.
Persistence sample size	Determines how sensitive the PCD is to the genuineness of a given change.

Table 3.1: PCD parameters

We use DPC to refer to $DPC_{squashed}$ for simplicity. $PerFactor$ is used to scale down the sigmoid function ($tanh()$) input data in order for the function to give sensible output, i.e., that represents the combined change degree on the range of $[-1, 1]$. Determining $PerFactor$ is dependant on the range of performance to be measured for the controller agent. We are only concerned when the final DPC is negative, i.e., the agent is not performing well while the local traffic pattern is changing. This is because an agent should not relearn unless its performance is adversely affected by the possible traffic pattern change. At this stage, DPC will have a negative value only if the incorporated MAR was negative as the result of CUSUM is always positive. Hence, we chose the final DPC value to be in the range of $[0, 1]$ where $DPC = 1 - abs(DPC_{[-1,0]})$. Moreover, the closer DPC is to 0, the more severe the negative change is, given the original $(MAR \times DPC)$ value. At a later stage, we detect a so called genuine change upon a situation where a sample of DPCs are persistently crossing a given junction change threshold (JCT) fixed for all signalized junctions. The sampling of DPC starts when a single DPC value crosses the JCT and continues until the so called persistence sample is ready, (i.e., its size is met). That sample's mean is then compared against the JCT where a genuine pattern change is detected if this sample mean crosses the JCT, however, this is dependent of the thresholding technique used.

3.3.2 Sensitivity and Parameters

The sensitivity of any change detection mechanism, i.e., how representative of the change it is and what constitutes a change for it, can naturally be controlled through different means. In PCD, a number of decisions determine the overall sensitivity, see Table (3.1). These are divided into two groups; the filtering parameters group and the squashing and thresholding group.

A given moving average filter is characterized by the sample size s. The larger s is, the less sensitive the output of that filter becomes to the input. We use three moving average filters that require a

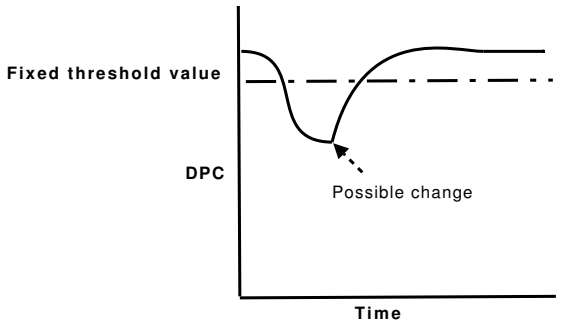


Figure 3.4: Fixed thresholding technique

preset sample size for traffic counts per lane, CUSUM filter outputs and agent rewards. In addition, the CUSUM filter requires preset k and n where k is the smaller moving sample size and n is the size of the whole moving sample. The values of k and n determine the sensitivity of the CUSUM filter and are typically determined empirically (explained below) as the choice depends on the nature of the input data.

On the other hand, *PerFactor* and the JCT are the two parameters that are closely related to determining the actual PCD sensitivity. *PerFactor* is fixed based on the range of data fed to the sigmoid function in order to help the function produce a representative output data on the sigmoid range $[-1, 1]$. In order to detect a genuine change, a given JCT is needed to determine the DPC that is considered the beginning of a possible change (discussed later). Moreover, the preset values of all the above mentioned sensitivity parameters can be determined empirically as they are domain dependant especially given the granularity of the original inputs.

A certain thresholding technique should be used in PCD. Figure (3.4) shows a fixed thresholding technique, which is a classical way were a defined threshold value remains static throughout the process of change monitoring. Other thresholding techniques can be used in PCD, for instance, dynamically changing thresholds but this is discussed as future work.

Regardless of the thresholding technique used, the notion of persistence is introduced as the criterion to distinguishing a genuine change. The latter is determined by comparing the mean of a DPC sample of a preset size (persistence sample size), collected after a single DPC value crosses the fixed JCT, against the JCT value. If the DPC sample's mean is lower than the JCT value, the change is declared genuine. The persistence sample size is also considered relevant to the sensitivity; the larger the sample the less sensitive the design is and vice versa.

By empirically in this section it is meant that a small scale simulation study, typically on single signalized junction, was carried out for the sole purpose of choosing suitable parameter values. A set

of candidate PCD parameter values, mainly for the first six parameters in Table (3.1), were evaluated and the resulting DPC is monitored throughout the simulation that includes different traffic patterns. The JCT and the persistence sample size parameters were consequently determined after analyzing the series of DPC values throughout the simulation.

3.3.3 Algorithm

The PCD process runs continuously as long as the agent is running. The main task for it is to determine if the agent needs to relearn upon the detection of a persistent change. The different sample sizes, $Perfactor$ and the JCT needed for the PCD process have to be determined empirically and are then initialized as common values for all signalized junctions. For a given junction, the PCD process can be described as in Algorithm (6). It continuously invokes the DPC calculation process (see Algorithm (5)) while detecting genuine changes.

The DPC calculation process updates the different samples needed by PCD and calculates a DPC value. By updating a sample, it is meant that a new reading is added to the sample which is in a sliding window form, so whenever the sample is full the oldest reading is removed and a new one is added. The DPC calculation process starts by updating the samples of traffic counts per lane and calculates the moving average for all. These averages are used to update the samples fed as input to the CUSUM filter. The CUSUM values are then calculated for all inputs and are used to update the CUSUM output samples. These samples are then passed to final moving average filter that stores the resulting sample average per lane identifier, (e.g., MA_CUSUM[Lane_ID]). At the same time, the agent reward sample is updated and its moving average is stored in MAR. In order to calculate the final DPC, steps $5a$ and $5b$ in Algorithm (5) that were discussed earlier in the design section are executed.

The PCD process starts by initializing the sizes of samples needed and other parameters such as $PerFactor$ and JCT. A boolean variable that determines when the PCD process is collecting a DPC sample after a given DPC has crossed the fixed JCT (DPC sampling for persistence status) is also initialized. The PCD then tries to obtain a DPC value, if it did, it checks whether this DPC crosses the JCT. If so, the DPC sampling for persistence status becomes true on the condition that the agent is not already learning. Consequently, while the DPC sampling for persistence status is true, the DPC sample for persistence is continuously updated until it meets the required sample size. The mean of that sample is then compared against the JCT and the agent is asked to relearn if the mean is equal to or less than the fixed JCT. In that case, the DPC sampling for persistence status becomes false and the DPC sample for persistence is cleared. If the agent is asked to relearn, it is passed the mean

Algorithm 5 PCD - calculate DPC

1. $calcDPC = false$
2. Update $SamplesofTrafficCountsPerLane$
3. If ($SamplesofTrafficCountsPerLane$ are full*)
   (a) Update $CUSUM\_InputSamplesPerLane$ using CalculateMovingAVG($SamplesofTrafficCountsPerLane$)
   (b) If ($CUSUM\_InputSamplesPerLane$ are full)
      i. Update $CUSUM\_OutputSamplesPerLane$ using CalculateCUSUM($CUSUM\_InputSamplesPerLane$) eq(3.1)
      ii. If ($CUSUM\_OutputSamplesPerLane$ are full)
         A. $MA_CUSUM[Lane_ID] \leftarrow$CalculateMovingAVG($CUSUM\_OutputSamplesPerLane$)
         B. Update $AgentRewardSample$ by querying for the last agent reward
         C. If ($AgentRewardSample$ is full)
            $MAR \leftarrow$CalculateMovingAVG($AgentRewardSample$)
            $calcDPC = true$
         D. EndIf
      iii. EndIf
   (c) EndIf
4. EndIf
5. If ($calcDPC$)
   (a) $DPC \leftarrow tanh((MAR \times mean(MA\_CUSUM))/\, PerFactor)$
   (b) $DPC \leftarrow (1 - abs(DPC_{[-1,0]}))$
6. EndIf

*By "full" it is meant that all the slots in the sample are occupied

Algorithm 6 The PCD process for a single signalized junction

1. Initialize
   (a) Sizes of all samples required, $Perfactor$ and JCT
   (b) $SamplingDPCforPersistence = false$
   (c) $DPC\_PersistenceSample.clear()$
   (d) $DPC = null$
2. $DPC \leftarrow$Algorithm (5)
3. While (agent is running) do
   (a) $If\ (FixedThresholding\ \&\&\ (DPC \leq JCT)\ \&\&\ Agent.isNotLearning())$
      i. $If\ (!\ SamplingDPCforPersistence)\ SamplingDPCforPersistence = true$
   (b) EndIf
   (c) $If\ (SamplingDPCforPersistence\ \&\&\ DPC\_PersistenceSample\ is\ not\ full^{\star})$
      i. Update $DPC\_PersistenceSample$
      ii. $If\ (DPC\_PersistenceSample\ is\ full\ \&\&\ (mean(DPC\_PersistenceSample) \leq JCT\ ))$
         A. $SamplingDPCforPersistence = false$
         B. Agent.relearn($true$, $mean(DPC\_PersistenceSample)$)
         C. $DPC\_PersistenceSample.clear()$
      iii. EndIf
   (d) EndIf
   (e) $DPC \leftarrow$Algorithm (5)
4. EndWhile

*By "full" it is meant that all the slots in the sample are occupied

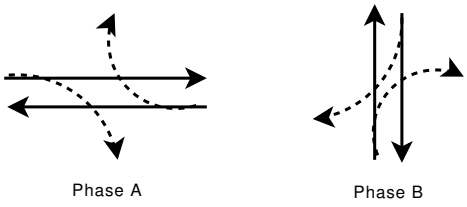


Figure 3.5: Two phases of a four-approach signalized junction

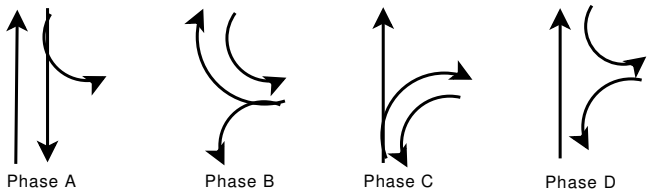


Figure 3.6: Complete set of phases for a T-shaped junction

of the DPC sample that determined the existence of a genuine change. Agent relearning is described in the Soilse and SoilseC design sections later in this chapter where new learning and action-selection parameter calculations based on the DPC sample mean are also presented.

3.4 Phases

An integral part of the control of a signalized junction is the specification of the available traffic phases. Phases can vary in their number and characteristics depending on the signalized junction's layout (i.e., geometry), pedestrian priority and other pure traffic engineering decisions. A given phase allows traffic on specified approaches to cross the junction towards permissible outgoing links. Phases are mutually exclusive in the sense that a given junction will only have a single phase active at a given time in order to avoid conflicting traffic. Here, we present some possible phase design choices and then specify how we define the set of phases we adopted in the Soilse approach.

A design comprising two phases for a four-approach signalized junction is presented in Figure (3.5). In this design, turning traffic (dotted arrows) is assumed to be infrequent and of less importance. Hence, turning traffic can wait for the opportunity to proceed when the absence of opposing traffic permits. Although this a straightforward design choice, it has clear safety issues given that the potentially risky choice of turning in any direction is solely the driver's responsibility.

Now consider a simpler T-shaped junction. A complete set of phases can be defined as the set of all phases representing all non-conflicting traffic directions at a given time, see Figure (3.6). This

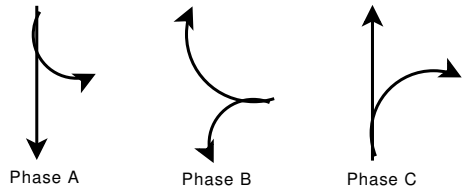


Figure 3.7: Simplistic set of phases for a T-shaped junction

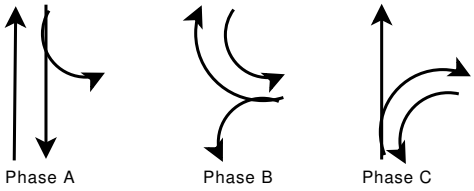


Figure 3.8: Concise set of phases for a T-shaped junction

design results in a large number of phases per signalized junction, which is an unfavoured result by traffic engineers.

On the other hand, a simplistic design would be to allow traffic only for a single approach to cross per phase, see Figure (3.7). However, this design choice misses the natural opportunity of letting non-conflicting traffic to cross. As a compromise, a phase design choice should result in a small number of phases and also serve non-conflicting traffic. This design choice eliminates certain phases from the complete set of phases by cross matching with the simplistic set, i.e., only phases in the complete set that include a simplistic phase are considered. This results in a so called concise set of phases, see Figure (3.8). The concise set hence comprises fewer phases than the complete set and still allow less restrictive phases compared to the simplistic set. Usually, traffic engineers favour selecting the minimum number of phases specifically engineered for better traffic flow on junctions. Consequently, we use the concise set of phases with our approach.

3.5 Signalized Junction Model - Soilse and SoilseC

Having described the PCD mechanism in Section (3.3), we now describe the design of both Soilse and SoilseC signalized-junction controller agents and show how PCD fits into the overall design. A Soilse agent is defined as an independent agent that controls a signalized junction using a policy reached by RL-based optimization in a responsive (see *Req*1) and adaptive (see *Req*2) manner. A SoilseC agent is defined as an agent that controls a signalized junction using a policy reached by RL-based

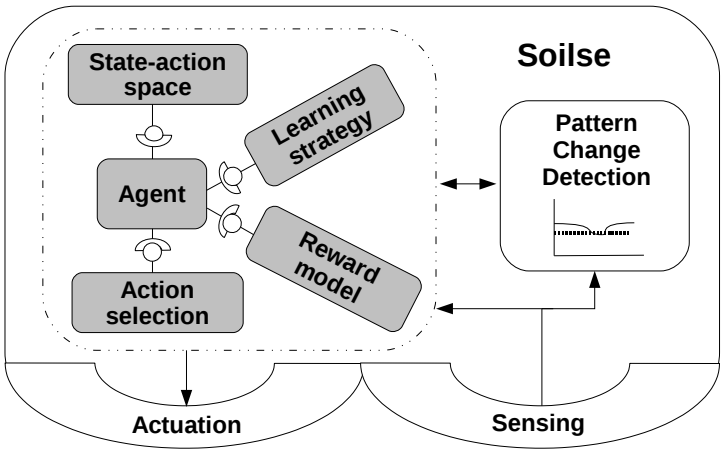


Figure 3.9: Soilse agent structure

optimization in collaboration (see *Req*4) with neighbouring SoilseC agents also in a responsive (see *Req*1) and adaptive (see *Req*2) manner. The responsiveness of both Soilse and SoilseC agents is supported by a PCD mechanism that detects genuine traffic pattern changes that adversely affect the agent's performance.

3.5.1 Soilse

A Soilse agent, see Figure (3.9), is composed of an RL agent and a PCD module. The RL agent comprises a representation of the environment, i.e., a state-action space, strategies for action selection and learning as well as a reward model. Actuation and sensing are provided through generic interfaces (see *Req*3).

The PCD module interacts with the RL agent by enquiring about the agent's performance, (i.e., rewards) as well as triggering relearning when required. It periodically polls the sensing interface for traffic counts per lane on the given junction in order to carry out the PCD process. If a genuine traffic pattern change is detected, the PCD mechanism passes the resulting DPC value to the RL agent. The DPC is used by the RL agent to calculate new learning parameters including both the learning and action selection strategies (assuming that the action selection strategy is not pure greedy).

Through the sensing interface, the RL agent is able to receive information about the environmental situation. Such information could be, but is not exclusively, the amount of traffic that crossed the junction during a given phase as well as the current traffic counts on a given phase's incoming approaches. The source of that information can vary from classical traffic cameras or inductive loops to recent FVD technologies. Consequently, the RL agent knows its current state and is able to select

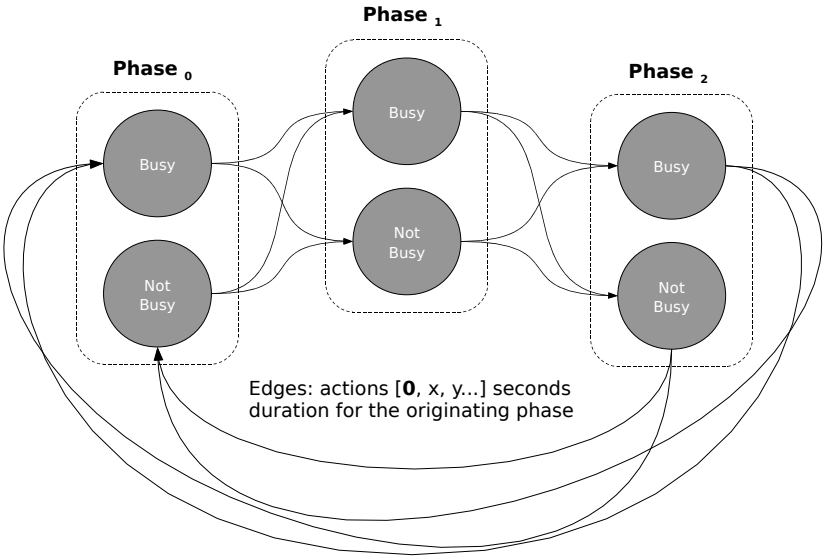


Figure 3.10: Soilse agent state-action space for a signalized junction with three phases

the next action using its action selection strategy, calculate a reward using a given reward model after the action is executed, update its policy using its learning strategy and, finally, update its new state. Moreover, the RL agent can change its environment through the actuation interface that allows, for instance, changing of the signalized junction's phase setting.

The learning strategy that is used in Soilse is a Q-Learning one. The choice of using Q-Learning stems from it being a well-established model-free off-policy RL strategy. It is a model-free approach in the sense that it does not require some a priori likelihood model for the actions that can be executed on the environment. It is also an off-policy RL strategy as it learns and updates the agent's knowledge even while taking actions that might prove to be non-optimal in the future (Abdulhai et al., 2003). Being an model-free off-policy learning strategy, as well as allowing for short period knowledge updating per action taken, Q-Learning is an ideal candidate for UTC optimization given the non-stationary nature of traffic (Abdulhai et al., 2003; Abdulhai & Pringle, 2003).

In terms of action selection, Boltzmann, ϵ-greedy and greedy are supported. The state-action space representation is based on an adaptive round-robin design of all concise phases and their possible different timings (see Section (3.2)). The state-action space varies in size depending on the number of phases available per signalized junction. Figure (3.10) depicts the state-action space of a signalized junction with three phases available.

In a Soilse agent, a given signalized junction's state-action space is modelled based on every available phase and its status, (i.e., busy/not busy). A given phase's status depends on all the incoming

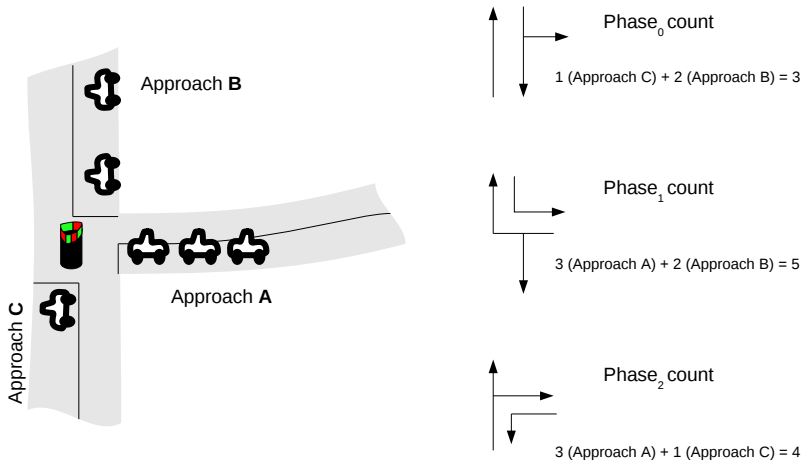


Figure 3.11: Phase traffic count - used to determine phase status

approaches of that phase, see Figure (3.11). A pair of a phase and its status is considered a state, (e.g., $s_y \implies (Phase_x\, is\, busy)$) in the model, see Figure (3.10). A given phase's status is determined by comparing the total number of vehicles within queueing range on its incoming approaches against a specific threshold value. A Soilse agent provides a number of actions, (i.e., candidate phase durations including a zero-second duration action) that could possibly be chosen in a given state. Given that we follow a round-robin style over n phases, after any action we take in any state of phase P_i, the next action will be in a state of phase $P_{(i+1)\, mod\, n}$ depending on local traffic conditions. The availability of a zero-second duration action allows the Soilse agent to skip unnecessary phases while exploring for policy optimization.

3.5.1.1 Local Reward Model

The design of any RL agent crucially relies on the reward model. The local reward model defines the optimization criteria pursued by a signalized junction. These criteria could be to minimize overall vehicle waiting time or number of stops, or to maximize throughput. However, in UTC the interrelation of metrics is inevitable, for example, reducing vehicle waiting time affects the number of stops and vice versa. Also, optimizing for increased throughput could affect both vehicle waiting time and number of stops. Consequently, the local reward model needs to be fair in capturing the effect of an executed action, for example, in capturing the number of waiting vehicles after a given action as well as the number of vehicles that have crossed the junction due to that action. Moreover, better local throughput should naturally lower vehicle waiting time as it encourages vehicle movement as well as lower the number of stops that vehicles suffer en-route to their destinations. As a result, we provide

a design for the local reward model that we refer to as R1 as follows:

R1: this reward model aims at capturing the traffic that has crossed the junction during a given phase duration and the remaining waiting traffic on all approaches on the junction. The reward will result in a negative reinforcement if a given action (timing) on a given phase results in more traffic waiting on the junction as a whole compared to the traffic that has crossed. Otherwise, it is a positive reinforcement. Consequently, this reward model optimizes for fair local throughput, which aims at enhancing local performance in terms of vehicle waiting time and number of vehicle stops.

$$R1 = (\, number\, of\, vehicles_{\, crossed} - number\, of\, vehicles_{\, waiting\, on\, the\, junction}\,)$$

R1 is motivated by the continuous (as opposed to discrete) form of reinforcements it naturally provides while capturing the possible negative effect that a given action might cause on the junction as a whole. Besides being fair to waiting vehicles, it also reinforces actions resulting in better traffic flow. R1's continuous nature is more informative to the learning process as opposed to a discrete nature, (e.g, positive reinforcement = 1 , negative reinforcement = -1) given the complex nature of the UTC optimization problem.

3.5.1.2 Relearning

The need for relearning stems from the requirements of responsiveness and adaptiveness (see $Req1, 2$). Under a non-stationary urban traffic environment, an RL agent with a sole policy learnt for a given traffic pattern cannot be expected to cope with new traffic patterns unless it relearns for each. Hence the need arises for relearning.

We propose an agent relearning strategy that is based on the DPC value passed from the PCD module when a genuine traffic pattern change is detected. The relearning procedure allows for responsiveness in the Soilse and SoilseC design. Based on that DPC value, new learning and action selection strategy parameters are calculated as follows:

Learning rate (α): given that the lower the DPC value, the more severe is the traffic pattern change that is adversely affecting the agent performance, a higher learning rate is needed in these cases in order to cope with the severe change. Since DPC has a range of $[0, 1]$ the mapping becomes simpler.

$$\alpha_{new} = (1 - DPC) \tag{3.3}$$

Epsilon in ϵ-greedy: like the new learning rate calculation, the need for higher exploration in case of a lower DPC value is needed. The epsilon in ϵ-greedy needs to be higher for the action selection to be more exploratory.

$$\epsilon_{new} = (1 - DPC) \tag{3.4}$$

Boltzmann temperature (τ): the temperature degree in a Boltzmann action selection strategy determines the degree of exploration and is proportional to the latter. The higher τ is, the more exploratory the agent using a Boltzmann action selection is. Hence, a new τ is calculated proportionally to the policy model size (number of state-action pairs) and the DPC value. The proportional relation can also be controlled using a so-called exploration factor (*ExpFactor*). The higher is the *ExpFactor*, the more weight is given to the policy model size relative to the DPC value.

$$\tau_{new} = (PolicyModelSize/DPC) \times ExpFactor \tag{3.5}$$

For example, if a state-action space has two states where each has two actions, the $PolicyModelSize$ for that state-action space would be $(1 \times 2) + (1 \times 2) = 4$.

In order to determine the duration of the relearning, a decay rate needs to be calculated per relearning parameter. The decay should be exponential using a generic formula (3.6) as we need the relearning parameters to decay in a manner that is proportional to their value but that reduces exploration gradually.

$$value_{new} = (e^{-(value_{decay\,rate}) \times time\,step}) \times value_{initial} \tag{3.6}$$

A natural logarithmic function is then used to calculate decay rates that are proportional to the DPC, (i.e., the higher is DPC value the faster the relearning/exploration should finish) but inversely proportional to the $PolicyModelSize$ and the *ExpFactor*, (i.e., the larger the $PolicyModelSize$ and the *ExpFactor* the slower the relearning/exploration should finish). The calculation is carried out as follows:

Under ϵ-greedy: when using ϵ-greedy as an action selection strategy, the decay rate for α and ϵ is common given their common relearning initial value, i.e., $(1 - DPC)$.

$$\alpha_{decay\,rate} = \epsilon_{decay\,rate} = \frac{log_e(1/(1 - DPC))}{PolicyModelSize \times ExpFactor} \tag{3.7}$$

Under Boltzmann: when using Boltzmann as an action selection strategy, a common decay rate based on $\alpha_{decay\,rate}$ and $\tau_{decay\,rate}$ is used.

$$\tau_{decay\,rate} = \frac{|log_e(1/\tau_{Initial})|}{PolicyModelSize \times ExpFactor}$$

Algorithm 7 Soilse initialization

Initialize α, $\alpha_{decay\,rate}$, γ, $policy \forall Q(s,a)$, $PCD.JCT$, $PCD.PersistenceSampleSize$

Set Initial State/Action $s_t \in S : \{all\,agent\,states\}$, $a_t \in A_t : \{all\,actions\,for\,s_t\}$

Set Action Selection $AS \in [Boltzmann, \epsilon - greedy, greedy]$

Initialize $((\tau, \tau_{decay\,rate}, common_{decay\,rate})\,or\,(\epsilon, \epsilon_{decay\,rate}))$ depending on AS

Set Local Reward Model $Rx \in Local\,Rewards$

$$common_{decay\,rate} = (\alpha_{decay\,rate} + \tau_{decay\,rate})/2 \tag{3.8}$$

The $\alpha_{decay\,rate}$ in equation (3.8) is calculated as in the situation under ϵ-greedy. The $common_{decay\,rate}$ is hence used for the decay of both τ and α in the Boltzmann case. The value of the $ExpFactor$ is determined empirically (described in Section (3.3.2)).

3.5.1.3 Soilse Algorithm

The Soilse agent uses Q-Learning as its learning strategy and can choose the action selection strategy to be Boltzmann, ϵ-greedy or greedy. All Soilse agents in a given deployment use the same action selection type. In Algorithm (8) the Soilse process is presented.

The Soilse agent starts by initializing the needed parameters relevant to its PCD, learning and action selection strategies (see Algorithm (7)). The agent then executes an action, receives its next state and calculates its local reward. The agent then uses its local reward for its policy update using Q-Learning and selects a next action using a given action selection strategy. The agent then updates its state and the action to take. Furthermore, the agent checks whether it was asked to relearn by its PCD, if so, it reparameterizes itself using the DPC value passed by the PCD. Naturally, the agent decays its learning and action-selection related parameters as long as it is exploring. The Soilse agent will continue exploration as long as its learning parameters have not reached their preset minimum values where exploitation then begins. The preset minimum values are when $\tau = 1$ and $\alpha \approx \epsilon \approx 0$.

3.5.2 SoilseC

Having detailed the design of a Soilse agent that operates independently, the collaborative version is now described. A SoilseC agent has the same design of a Soilse agent with an added element that allows for collaboration, i.e., an advertisement strategy, see Figure (3.12). In addition, a collaborative reward model is used as opposed to the local reward model in Soilse agents. That collaborative reward

Algorithm 8 Soilse process

Initialize Soilse Algorithm (7)

While (*Soilse is running*)

Execute a_t ; **Receive** s_{t+1}

Calculate Reward r_{t+1} based on Rx

$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha\left[r_{t+1} + \gamma\, max_a Q(s_{t+1}, a) - Q_t(s_t, a_t)\right]$

Select $a_{t+1} \in A_{t+1} : \{all\, actions\, for\, s_{t+1}\}$ using AS

$s_t \leftarrow s_{t+1}$, $a_t \leftarrow a_{t+1}$

If (*relearn*) #Relearn status is updated by PCD Algorithm (6) where it passes the DPC to be used in reparameterization

Reparameterize(AS, DPC) #Algorithm (9)

EndIf

If (*Soilse.exploration* == *true*)

Decay α and (τ or ϵ depending on AS) using eq(3.6)

EndIf

EndWhile

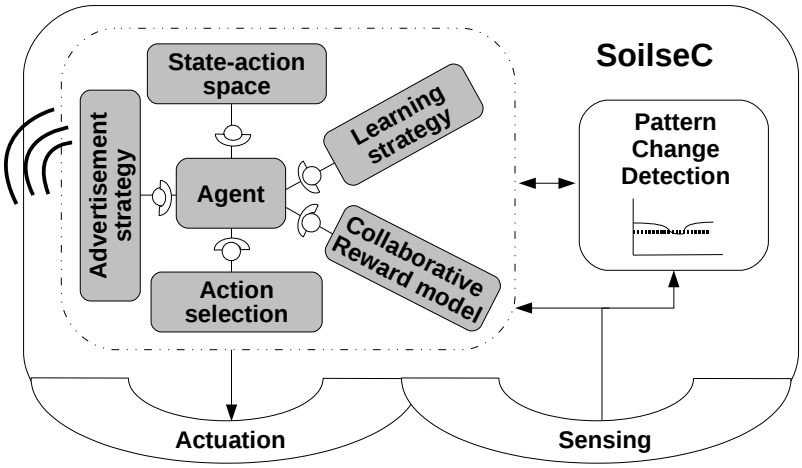


Figure 3.12: SoilseC agent structure

model includes an implicit local reward model similar to the Soilse agent design but also allows for the incorporation of exchanged information.

The advertisement strategy determines the collaboration mode for each SoilseC agent. The mode describes which other SoilseC agent(s) to send and receive to/from. Exchanged information provides

Algorithm 9 Reparameterize per action selection strategy

Switch(AS)

case(Boltzmann):{

$\tau \leftarrow eq(3.5)$; $\alpha \leftarrow eq(3.3)$

$\alpha_{decay\,rate} \leftarrow \tau_{decay\,rate} \leftarrow common_{decay\,rate}\, eq(3.8)$

}break

case(ϵ-greedy):{

$\alpha \leftarrow (3.3)$; $\epsilon \leftarrow eq(3.4)$

$\alpha_{decay\,rate} \leftarrow \epsilon_{decay\,rate} \leftarrow eq(3.7)$

}break

case(greedy):{

$\alpha \leftarrow eq(3.3)$

$\alpha_{decay\,rate} \leftarrow eq(3.7)$

}break

EndSwitch

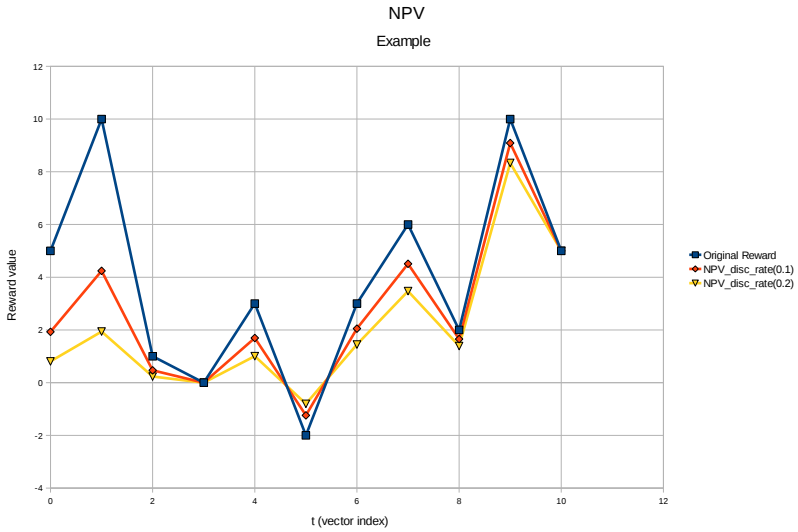


Figure 3.13: NPV example

a metric for the sending agent's recent performance. Such information includes a series of rewards ordered by age. The older the reward value is, the less important it is. Exchanging information occurs at a predefined frequency for all collaborating SoilseC agents ($CollFreq > 0$). The exchanged rewards are discounted using a Net Present Value (NPV) (Lin & Nagalingam, 2000) inspired Equation (3.9) that is a well-known method used in economics for discounting a series of values based on age. The NPV equation diminishes the significance of older rewards based on a given $disc\_rate$ value. An r_t is the reward obtained at index t in the exchanged reward vector. The most recent reward has the highest t value while the first has $t = 0$, hence, $0 \leq t < rv\_size$ and rv_size is the reward vector size. See Figure (3.13) for an NPV example with two different $disc\_rate$ values.

$$NPV(r_t) = \frac{r_t}{(1 + disc\_rate)^{(rv\_size-(t+1))}} \tag{3.9}$$

The collaborative reward model (described in subsection (3.5.2.2)) is able to use the cached, (i.e., simply stored) exchanged rewards and discount them according to the proposed procedure from the advertisement strategy, i.e., the NPV in this case. Determining the values of the $CollFreq$ and the NPV's $disc\_rate$ parameters is a matter of design choice that is directly related to the nature of the collaboration sought. The more frequent the collaboration is, the less reward history is exchanged and vice versa. Hence, a value is needed that depends on to the action durations available to the agents as a reward is received after each action. For example, if the maximum action duration available is 40 *seconds*, then a collaboration frequency of 240 *seconds* would guarantee at least a reward history of size 6 if the agent would choose to execute the maximum duration action continuously.

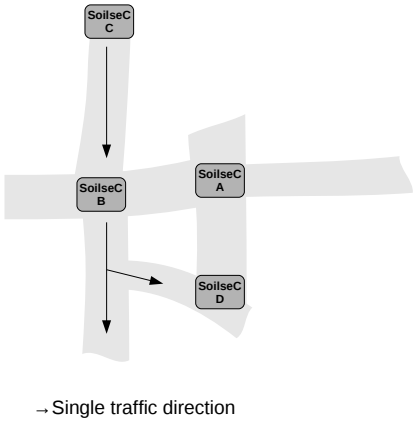


Figure 3.14: Neighbours example

3.5.2.1 Neighbours

The set of neighbours for any SoilseC agent comprises all one-hop SoilseC agents whether upstream or downstream ignoring any non-signalized junctions in the way. Consequently, the advertisement strategy suggests three possible modes of collaboration.

Mode one allows for a given SoilseC agent to receive information only from upstream neighbours and send information only to the downstream ones.

Mode two allows for a given SoilseC agent to receive information only from downstream neighbours and send information only to the upstream ones.

Mode three allows for a given SoilseC agent to send and receive information from upstream and downstream neighbours.

For example, consider the traffic network layout in Figure (3.14). The neighbourhood of SoilseC agents could be described as in Table (3.2). Depending on the the mode of collaboration, a given SoilseC agent selects other SoilseC agents from its neighbourhood which it sends to, receives from information or both. In the SoilseC design, information about the distance between signalized junctions is not modelled in the agents given that RL, by its nature, is an unsupervised learning approach. Adding such information is considered a form of supervision while SoilseC agents are expected to learn the dynamics of the shared environment through information exchange and local interaction.

Neighbours/SoilseC agent	A	B	C	D
Upstream	B,D	A,C	-	A,B
Downstream	B,D	A,D	B	-

Table 3.2: SoilseC neighbourhood

3.5.2.2 Collaborative Reward Model

The collaborative reward model is formed from the combination of a local reward model and exchanged information. Such a model makes use of the advertisement strategy by accessing received information. Assuming that the received information is a vector of rewards per sender, the collaborative reward model uses the NPV method in the advertisement strategy in order to discount the received/cached rewards by age. A normalization procedure follows per number of senders and the size of reward vectors for each.

$$CollStatus = \frac{\sum_{\forall n \in sending\ neighbours} \frac{\sum_{t=0}^{t=rv\_size_n - 1} NPV_n(r_t)}{rv\_size_n}}{number\ of\ sending\ neighbours} \tag{3.10}$$

$$r_{coll} \leftarrow (r_{local} + CollStatus) \tag{3.11}$$

Consequently, a single value (*CollStatus*, see Equation (3.10)) denoting the overall recent performance of all sending SoilseC agents is calculated. The incorporation step is achieved by adding *CollStatus* to the current local reward value, see Equation (3.11).

3.5.2.3 SoilseC Algorithm

The SoilseC algorithm is similar to the non-collaborative Soilse algorithm with the addition of information exchange depending on the mode and the incorporation of this information locally. Algorithm (11) describes the SoilseC process.

The SoilseC agent starts by initializing the required parameters relevant to its PCD, learning and action selection strategies (see Algorithm (10)). The agent then executes an action, receives its next state and calculates its local reward. The latter is added to its local reward history which is used for its performance information exchange. If it is time for collaboration (depending on a predefined frequency) and the local reward history is not empty, the agent advertises its local reward history to a predefined set of neighbours. The agent then calculates its collaborative reward, which is used for its policy update using Q-Learning. It then selects the next action using the given action selection strategy and updates its state and the action to take. Furthermore, the agent checks whether it

Algorithm 10 SoilseC initialization

Initialize

$CollFreq$, $NPV.disc\_rate$, $LocalRewardsHist$, α, $\alpha_{decay\,rate}$, γ, $policy\,\forall Q(s,a)$
$PCD.JCT$, $PCD.PersistenceSampleSize$

Set Initial State/Action $s_t \in S : \{all\,agent\,states\}$, $a_t \in A_t : \{all\,actions\,for\,s_t\}$

Set Action Selection $AS \in [Boltzmann, \epsilon - greedy, greedy]$

Initialize $((\tau, \tau_{decay\,rate}, common_{decay\,rate})\,or\,(\epsilon, \epsilon_{decay\,rate}))$ depending on AS

Set Collaborative Reward Model CRx #Implicitly assigns a local reward model

Choose Collaboration Mode CM #See Subsection (3.5.2.1)

Build Neighbourhood $N : \{SendToN\} \cup \{ReceiveFromN\}$ given CM

was asked to relearn by its PCD, if so, it reparameterizes itself using the DPC value passed by the PCD. Naturally, the agent decays its learning and action-selection related parameters as long as it is exploring. The SoilseC agent will continue exploration while its learning parameters have not reached their preset minimum values where exploitation then begins. The preset minimum values are when $\tau = 1$ and $\alpha \approx \epsilon \approx 0$.

3.6 Summary

This chapter describes the design for a non-parametric PCD technique that can be deployed locally. The PCD presented does not rely on any a priori traffic model but rather detects changes in traffic and quantifies the change, i.e., the DPC. Two types of RL-based UTC agents were presented, Soilse and SoilseC, where the latter is collaborative. These agents make use of the DPC value upon a given change in order to calculate their relearning parameters. Occurrences of such a change are determined depending on a fixed thresholding technique.

Both the Soilse and SoilseC agent designs follow an adaptive round-robin RL optimization scheme in the core. This allows for a near optimal setting of a concise set of phases to be reached where unsuitable phases for a given traffic pattern can be skipped and others can be assigned adequate timings. Soilse and SoilseC agents are able to use different reward models, however, a reward model that balances between increasing local throughput and the number of waiting vehicles is provided. They use Q-learning as the learning strategy and could run using either Boltzmann, ϵ-greedy or greedy as an action selection strategy. The three different types of action selection strategies are assessed

Algorithm 11 SoilseC process

Initialize SoilseC Algorithm (10)

While ($SoilseC\,is\,running$)

Execute a_t ; **Receive** s_{t+1}

Calculate Local Reward $r_{local} \leftarrow CRx.CalcLocalReward()$;
LocalRewardsHist.push_back(r_{local})

If ((((t + 1) $mod\,CollFreq$) == 0) && ($LocalRewardsHist.empty() \neq true$))

Advertise (LocalRewardsHist, SendToN)

LocalRewardsHist.clear()

EndIf

$CollStatus \leftarrow eq(sending\,neighbours \leftarrow ReceiveFromN)$(3.10)

$r_{coll} \leftarrow (r_{local} + CollStatus)$ $\#eq$(3.11)

$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha\,[r_{coll} + \gamma\,max_a Q(s_{t+1}, a) - Q_t(s_t, a_t)]$

Select $a_{t+1} \in A_{t+1} : \{all\,actions\,for\,s_{t+1}\}$ using AS

$s_t \leftarrow s_{t+1}$, $a_t \leftarrow a_{t+1}$

If ($relearn$) #Relearn status is updated by PCD Algorithm (6) where it passes the DPC to be used in reparameterization

Reparameterize(AS, DPC) #Algorithm (9)

EndIf

If ($SoilseC.exploration == true$)

Decay α and (τ or ϵ depending on AS) using eq(3.6)

EndIf

EndWhile

in the evaluation Chapter (5) where also their suitability for UTC optimization under non-stationary urban traffic is discussed. In a SoilseC setting, neighbouring agents can collaborate following a common advertisement strategy and depending on one of the three supported modes which define senders and receivers (see *Req*4). Such a collaboration has shown promising results in terms of better global performance by taking into account neighbours' performance (Salkham et al., 2008). A collaborative reward model in the SoilseC agent uses the advertisement strategy to calculate a metric for sending neighbours status through discounting and normalizing communicated rewards.

Moreover, this chapter provided a decentralized RL scheme through which adaptive and responsive RL-based UTC agents can be deployed. Adaptiveness (see *Req*2) is provided through learning in a fair round-robin state-action space design that is based on a concise set of phases per RL agent. Concise phases and their timing are used as the means to control the setting of a traffic light given their coarse granularity that allows for a more compact state-action space. Responsiveness (see *Req*1) is supported through relearning based on new parameters calculated relative to the DPC value resulting from the non-parametric PCD technique upon a genuine local traffic pattern change. Moreover, Soilse and SoilseC including the PCD satisfy *Req*3 by not relying on a specific source of traffic sensor information but rather exposing generic sensing and actuation interfaces.

Chapter 4

Implementation

This chapter describes the implementation of our Collaborative Reinforcement Learning (CRL) framework which is a generic C++ framework that allows for the instantiation of CRL-based applications. It also describes the AgentsGenerator library that is used to create a Soilse or a SoilseC agent for each signalized junction and define the interface with the UTC simulator. The latter is described in Chapter (5).

4.1 The CRL framework

The use of a framework enables us to experiment with different application designs in a more structured and flexible manner. The CRL framework is a C++ library that provides the programmer with all the components needed to build an RL application, e.g., agents, learning strategies, action selection strategies, states, actions, Markov Decision Process (MDP) representation and model. By model in the CRL framework we mean the structure in which the learnt values are stored and indexed by some key, for instance, a key in our implementation takes the form of ($state\_ID$, $action\_ID$) if we are using Q-Learning. A $KeyValuePair$ is described as in the structure $pair < Key*, Key\_Value >$. A high-level class diagram for the CRL framework is presented in Figure (4.1).

The framework also supports collaborative application development by providing, in addition to the common RL application needs, a feedback or an advertisement strategy, neighbourhood management and caching. In that case, the model used has caching support for the information communicated from neighbouring agents. A certain advertisement strategy followed by every CRL agent supports agents in updating their local knowledge from their neighbours and decides on what information should be communicated to them. The remainder of this section describes the constituents of the

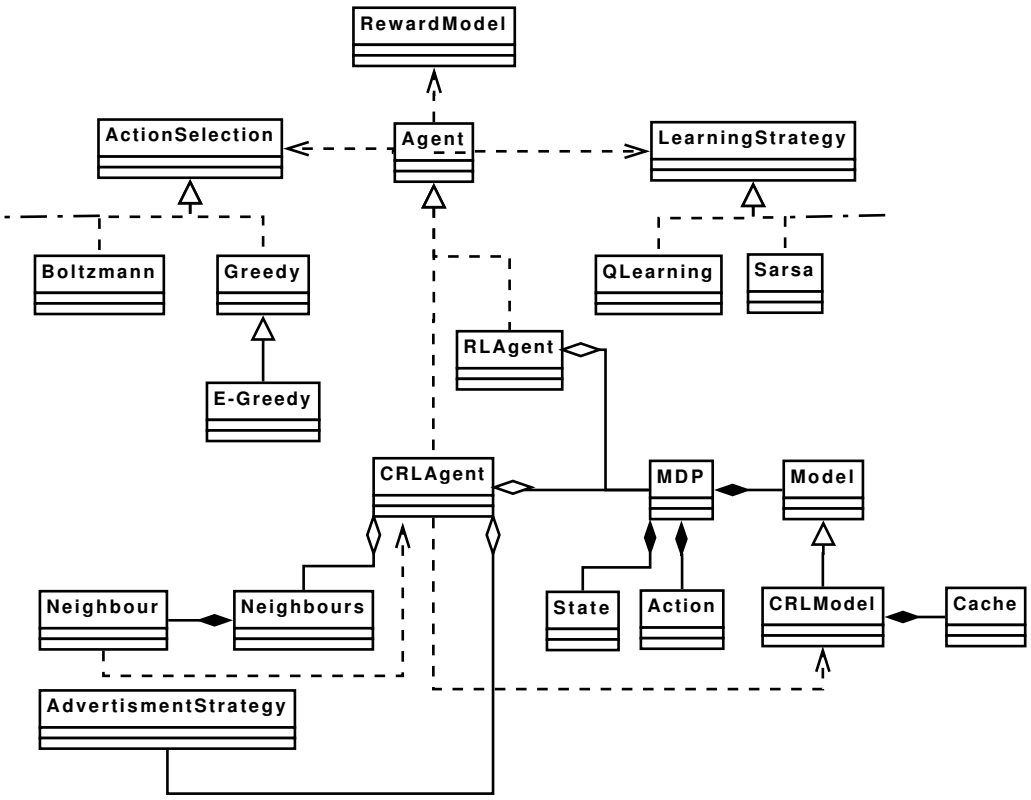


Figure 4.1: The CRL framework high-level class diagram

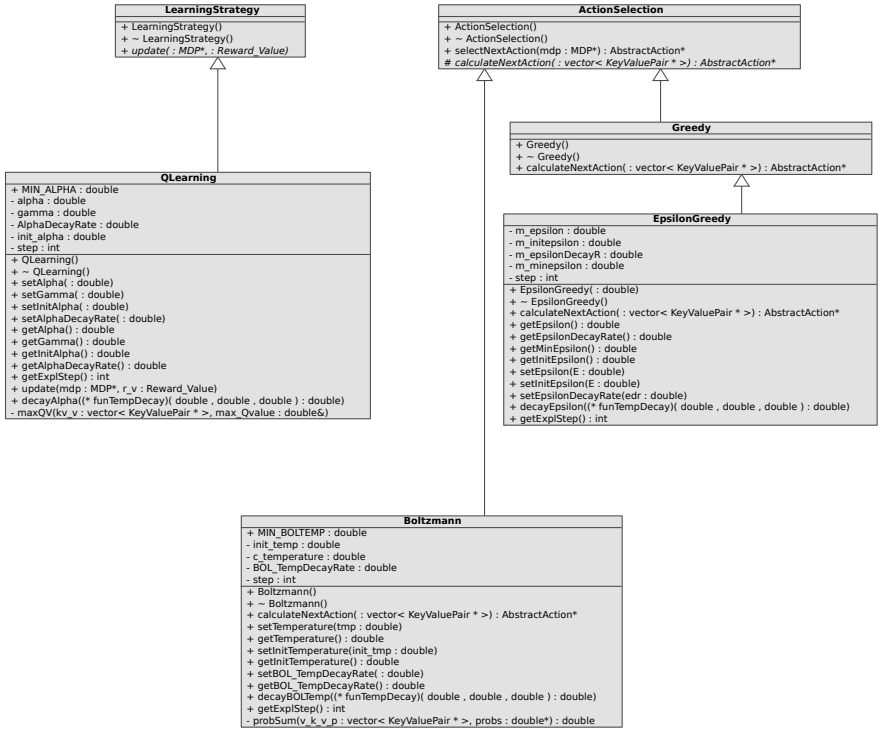


Figure 4.2: LearningStrategy and ActionSelection classes

CRL framework.

4.1.1 LearningStrategy

The LearningStrategy class provides the interface to which any learning strategy must implement. A Q-learning implementation is provided in the CRL framework but it is possible to add other strategies as needed, e.g., SARSA. The LearningStrategy class, see Figure (4.2), mainly specifies that any implementation of a given learning strategy must provide an *update* function which updates the agent policy.

The QLearning class inherits from the LearningStrategy class and implements the update pure virtual function exposed by the LearningStrategy class to suit the Q-learning logic, see Listing (4.1). It also has a number of additional functions and variables needed for its operation as can be seen in Figure (4.2).

```
void QLearning ::update(MDP* mdp, Reward_Value r) {

State * current_State  = mdp->getCurrentState();
State * next_State  = mdp->getNextState();
AbstractAction * current_Action = mdp->getCurrentAction();
Model * current_Model = mdp->getModel();
double max_Qvalue = 0.0;

/*Find the maximum Q-value (over the possible actions) for the next state*/
vector<KeyValuePair*> ns_a_qvalue = mdp->constructKeyValuePairs(next_State);

if (! ns_a_qvalue.empty() ) {

/*Get the maximum Q-value*/
maxQV(ns_a_qvalue, max_Qvalue);
vector<Key*> cs_ca_key;

/*Only needs one pair to be retrieved*/
Key cs_ca_k (current_State,current_Action);
cs_ca_key.push_back(&cs_ca_k);
vector<KeyValuePair*> current_s_a_qvalue = current_Model->getValues(cs_ca_key);

/*Main update rule*/
current_s_a_qvalue[0]->second +=
(alpha * (r + (gamma*max_Qvalue) - current_s_a_qvalue[0]->second));

        }
}
```

Listing 4.1: Q-learning update function

4.1.2 ActionSelection

Analogous to the LearningStrategy class, the ActionSelection class provides the interface, see Figure (4.2), with which any implemented action selection strategy should conform. Three types of action selection strategies are provided by the CRL framework, i.e., Boltzmann, greedy and ϵ-greedy. The latter is a special case of the greedy action selection and hence is a subclass of greedy.

The Boltzmann class implements the *calculateNextAction* pure virtual function exposed by the ActionSelection class. The *selectNextAction* function typically calls the *calculateNextAction* for the next set state. The Boltzmann *calculateNextAction* implementation is shown in Listing (4.2) as it shows the vital functional part of any action selection strategy.

```
AbstractAction* Boltzmann::calculateNextAction(vector<KeyValuePair*> v_k_v_p){

double * p = new double[(int)v_k_v_p.size()];
double sum = probSum (v_k_v_p, p);
double ran_num = ((double)rand() / RAND_MAX);
double normalised_p = 0.0 ;

for (int i = 0 ; i < (int)v_k_v_p.size() ; ++i) {
        /*Normalise and accumulate the probabilities*/
        normalised_p += (p[i] / sum);
        if (normalised_p >= ran_num) {
                delete[] p;
                return (v_k_v_p[i]->first->getAction());
        }
}

/* Return some action as a default behaviour*/
return ( (v_k_v_p.back())->first->getAction());
}

double Boltzmann::probSum (vector<KeyValuePair*> v_k_v_p, double * p) {

int p_size = (int)v_k_v_p.size();
double q = 0.0, sum = 0.0;

for (int i = 0; i < p_size ; ++i) {
        q = v_k_v_p[i]->second;
        /*Calculate Boltzmann's factor*/
        p[i] = exp( q / c_temperature );
        sum += p[i];
}
return sum ;
}
```

Listing 4.2: Boltzmann calculateNextAction

The Greedy and EpsilonGreedy classes also implement the *calculateNextAction* function, see Listing (4.3).

```
AbstractAction* Greedy::calculateNextAction(
vector<KeyValuePair*> v_k_v_p){

double max_q_value = 0.0;
int max_q_index = 0 ;
int p_size = (int)v_k_v_p.size();

for (int i = 0; i < p_size ; ++i) {
        if( i == 0 ) {
                max_q_index = i;
                max_q_value = v_k_v_p[i]->second;
    }else if( max_q_value < v_k_v_p[i]->second ) {
                max_q_index = i;
                max_q_value = v_k_v_p[i]->second ;
        }
}

return (v_k_v_p[max_q_index]->first->getAction());
}

AbstractAction* EpsilonGreedy::calculateNextAction(
vector<KeyValuePair*> v_k_v_p){

double r = ((double)rand() / RAND_MAX);
int index = 0;
int size = 0 ;
if( r >= getEpsilon() ) {
        return Greedy::calculateNextAction( v_k_v_p );
 }
else {
        size = (int)v_k_v_p.size() ;
        index = rand() % size ;
        return (v_k_v_p[index]->first->getAction());
 }
}
```

Listing 4.3: Greedy and EpsilonGreedy calculateNextAction

4.1.3 RLAgent

The most basic composition that can be instantiated from the framework is an RLAgent, see Figure (4.4). It combines the needed elements in order to have a functional RL agent that uses some action selection and learning strategies including a Model and an MDP representation of the environment. It is also associated with a RewardModel for a given optimization criteria. The RLAgent class implements

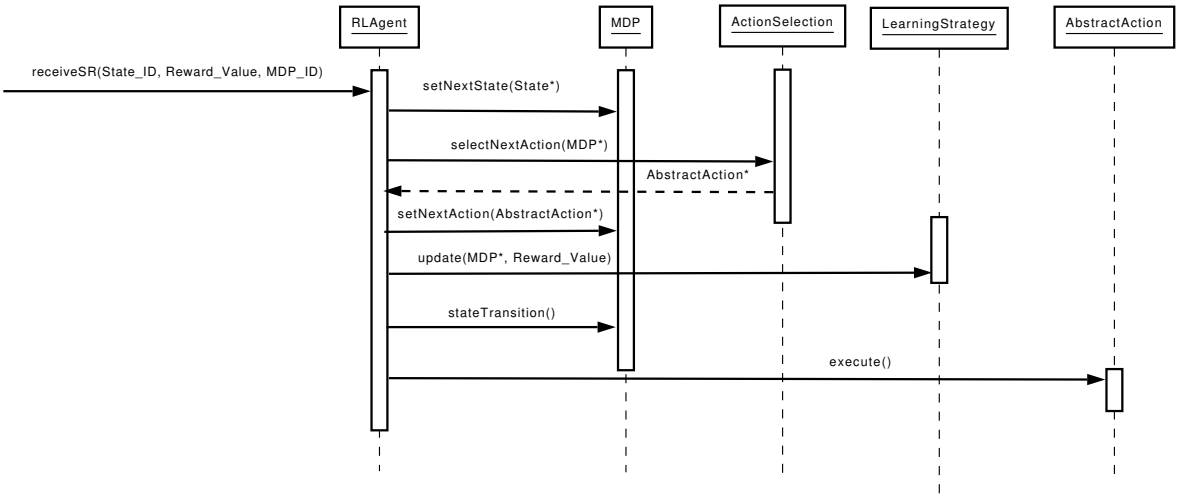


Figure 4.3: RLAgent receiveSR function sequence diagram

an Agent interface that defines the basic structure of an agent. The Agent class requires that a *virtual* *receiveSR* function is implemented where it also allows for the reward model and the learning and action selection strategies to be set. See Figure (4.3) for a typical *receiveSR* behaviour.

The RLAgent cannot function until its implementer builds an MDP for its environment and initializes its Model as well as customizes a given RewardModel. The MDP class provides a structure (see Listing (4.4)) to store the desired state-action space and exposes a pure virtual function *constructStateActionSpace*() that has to be implemented in order to instantiate a given MDP. It also keeps track of the current action and state as well as the next ones given that they are needed for the learning process.

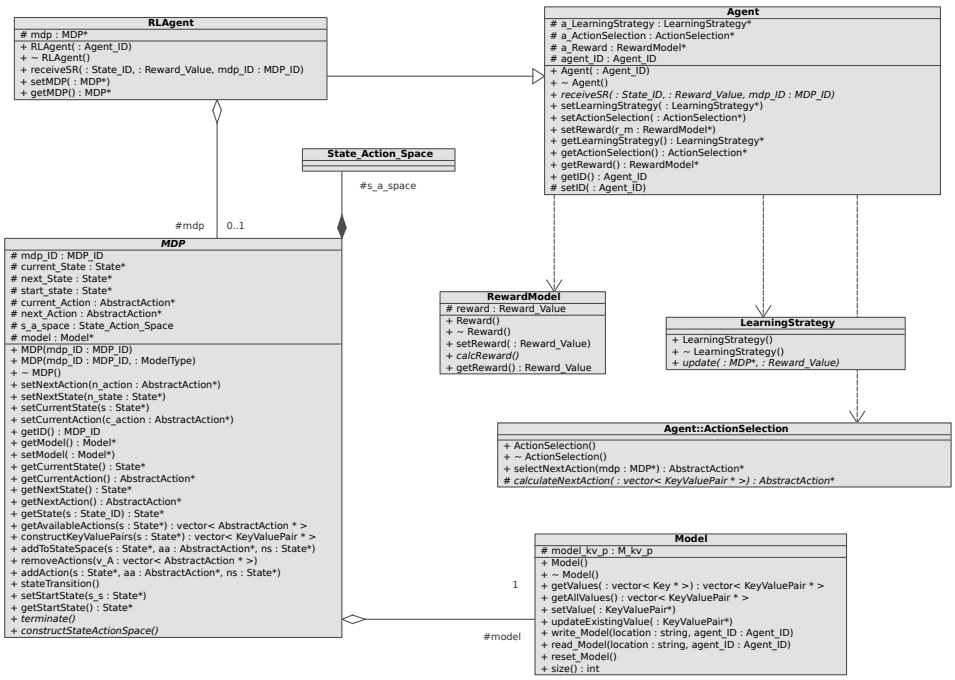


Figure 4.4: RLAgent class diagram

```
struct compare_states {
        bool operator()(State* s1, State* s2) const
        {
                return (s1->getID() < s2->getID());
        }
};
struct compare_AA {
        bool operator()(AbstractAction* aa1, AbstractAction* aa2) const
        {
                return (aa1->getActionID() < aa2->getActionID());
        }
};

typedef map< State*, map< AbstractAction*, vector<State*>, compare_AA >,
compare_states > State_Action_Space;
```

Listing 4.4: State-action space structure in the MDP class

The Model class provides a structure to hold the values associated with all possible state-action space keys. The subclass implementing an RLAgent has the responsibility to build/instantiate its Model. It also has to provide an implementation for the pure virtual function *calcReward*(). The implementation determines the reward calculation logic depending on a certain optimization criteria.

4.1.4 CRLAgent

The basic CRLAgent is similar in structure to the RLAgent with the addition of a neighbourhood management class, namely, Neighbours, an AdvertisementStrategy class and a Model with a Cache, namely, the CRLModel. Figure (4.5) represents the CRLAgent class and its relation to the other constituents.

The main function that the CRLAgent implements is the *receiveSR* where it specifies the steps taken upon receiving a new state and a given reward value. The sequence diagram in Figure (4.6) clarifies the process.

The Cache class supporting the CRLModel holds communicated information per neighbouring agent. This information will typically be the CRLModel contents of neighbours but could be any other kind of information sent from those neighbours. This is true as long as the information sent is in the form of a vector of $KeyValuePair$ structures. For example, a neighbour can send a group of its recent reward values with a key representing their order in time. Furthermore, the AdvertisementStrategy class provides the basic sending and receiving functionality. It also allows for the customization of the

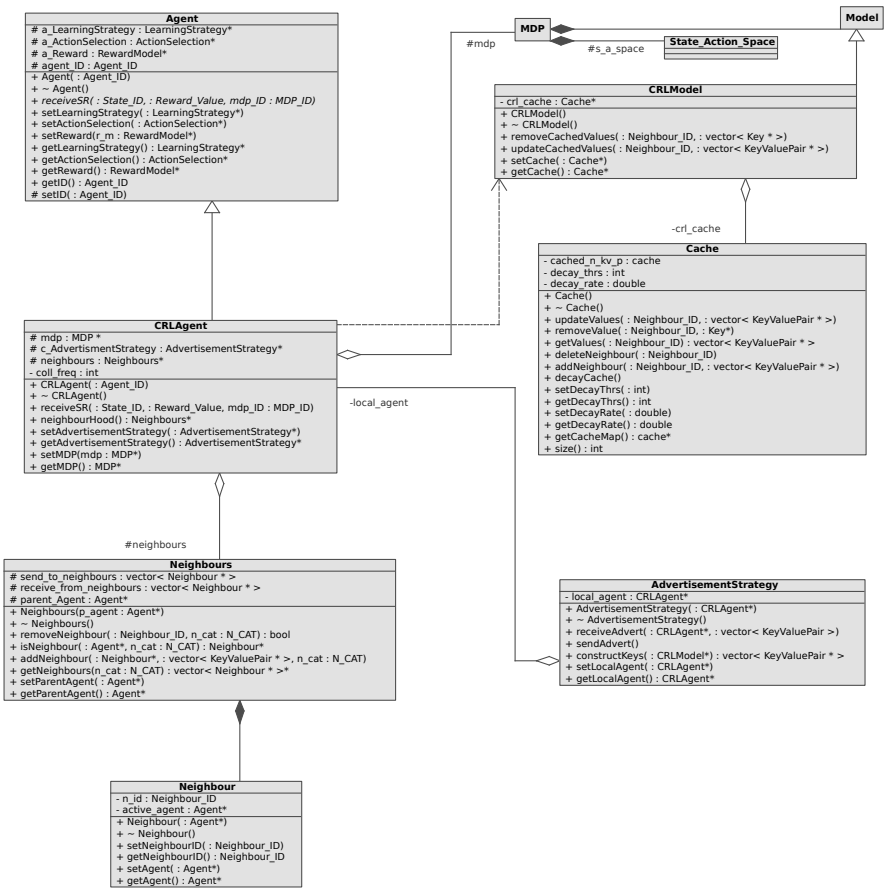


Figure 4.5: CRLAgent class diagram

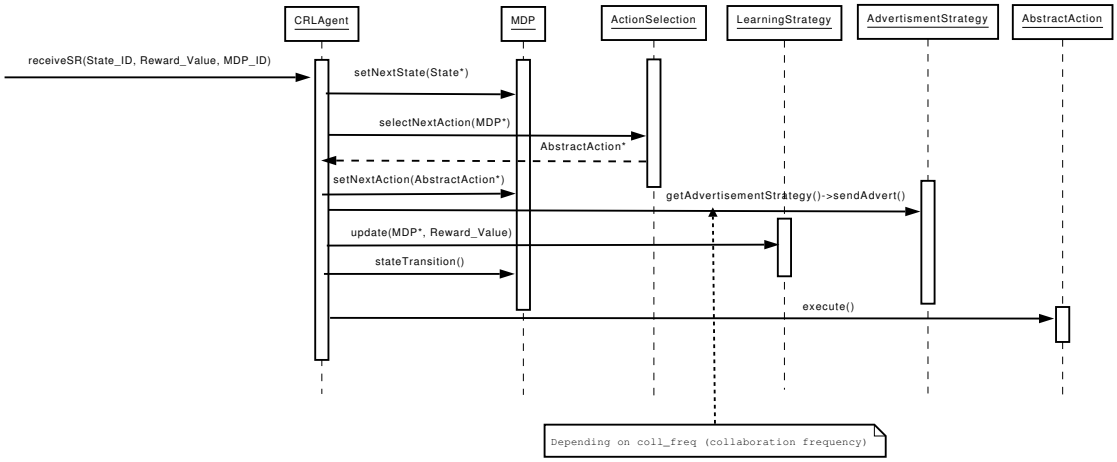


Figure 4.6: CRLAgent receiveSR function sequence diagram

constructKeys virtual function in order to determine what should be communicated to the neighbours. The default *constructKeys* implementation sends the entire cache contents unaltered.

4.2 Soilse and SoilseC Agent Generator

The Soilse and SoilseC agent-generator is a customized instance of the CRL framework with the addition of the PCD technique. It contains a set of classes that inherit and implement from different CRL framework classes. It also provides an interface class that acts as the connecting layer between the generated agents and the UTC simulator, i.e., the Sim_ENV class. Figure (4.7) presents the class diagram of the Soilse and SoilseC agents generator.

Although it is not explicitly shown in the agent-generator class diagram that certain classes inherit from the CRL framework, the relation is presented in Figure (4.8).

The high-level relation between the Soilse and SoilseC agent-generator, the UTC simulator and the CRL framework is depicted in Figure (4.9). The agent-generator instantiates a Soilse or a SoilseC agent (depending on the type of the agent-generator) for each signalized junction. These agent subclasses are an implementation for the RLAgent and CRLAgent classes in the CRL framework. The agent-generator enquires about the signalized junctions and their phases through the Sim_ENV class in order to use this information in the instantiation of Soilse or SoilseC agents. It also uses the provided learning and action selection strategies from the CRL framework in order to create such for each Soilse or SoilseC agent. Once the Soilse or SoilseC agents are created they can then interact with the simulator directly through the Sim_ENV interface.

4.2.1 PCD

The PCD implementation is encapsulated in a class that keeps track of the samples needed for its detection mechanism functionality. It provides the associated Soilse or SoilseC agent with a function to monitor the DPC. This function, namely, *monitorPatternChange*() tests the current DPC against a predefined threshold and checks its persistence if it crosses that threshold. Upon detection of a persistent DPC, the monitoring function returns a newly calculated DPC to the Soilse or SoilseC agent that is used to determine its relearning parameters. Listing (4.5) presents the monitoring function.

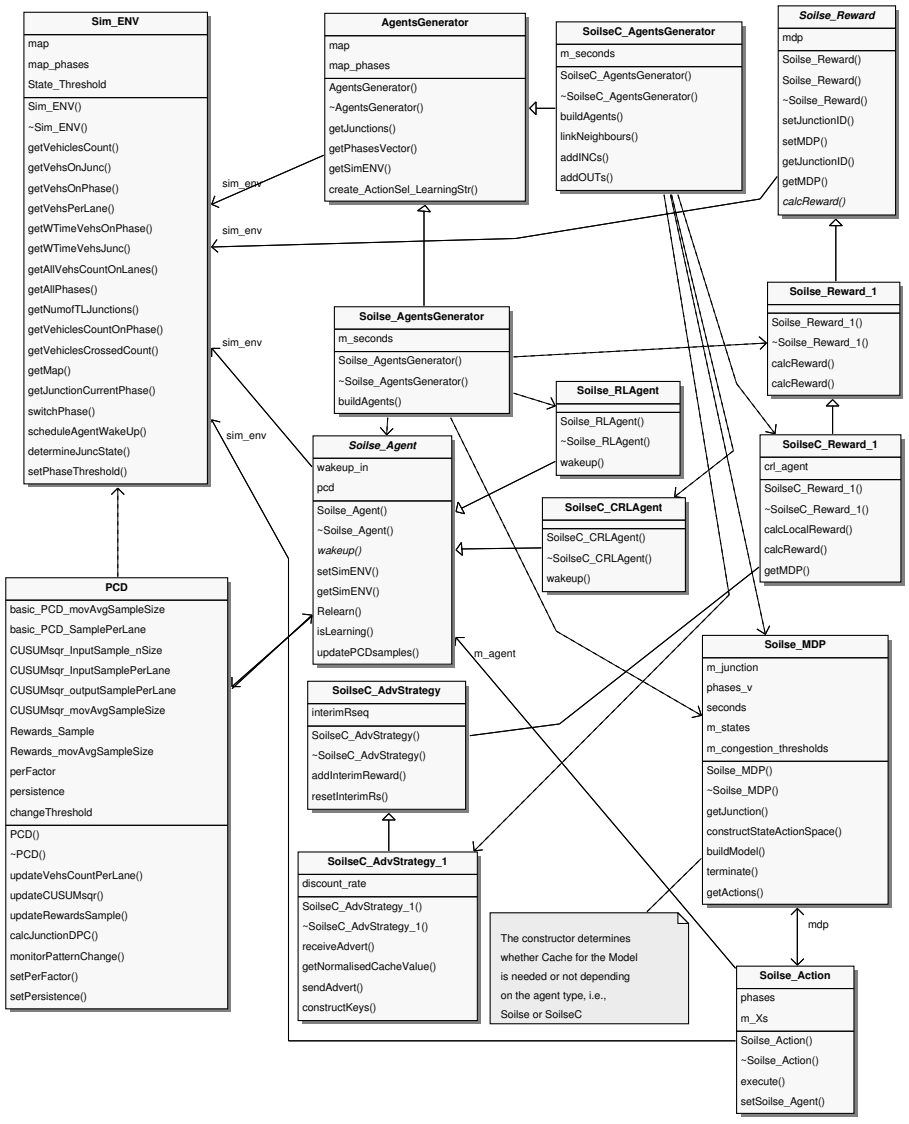


Figure 4.7: Soilse and SoilseC agents generator class diagram

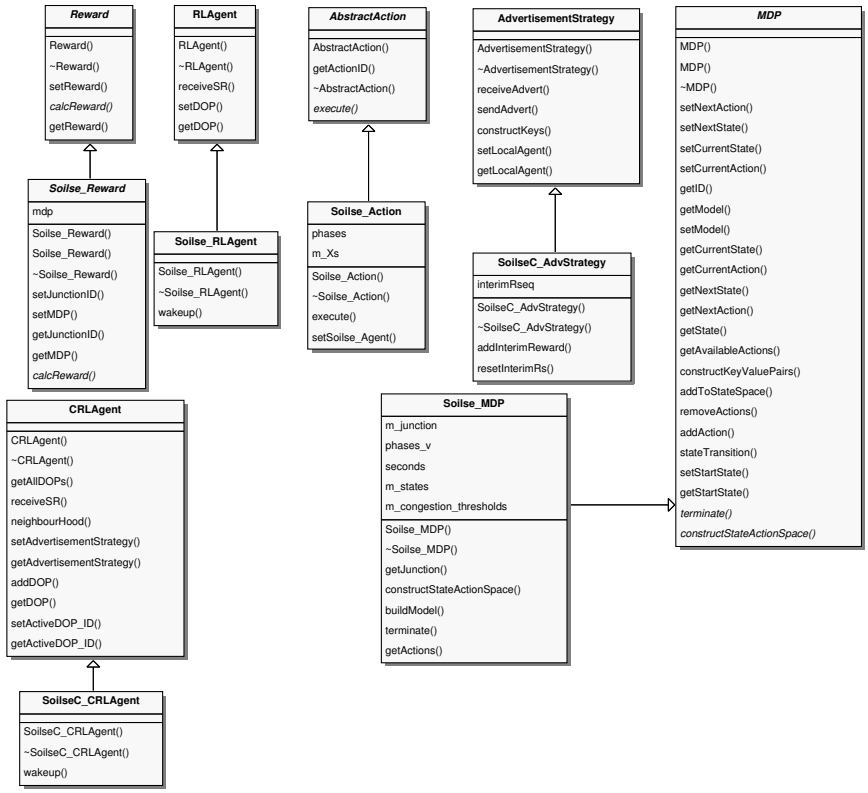


Figure 4.8: Soilse and SoilseC classes relation to the CRL framework classes

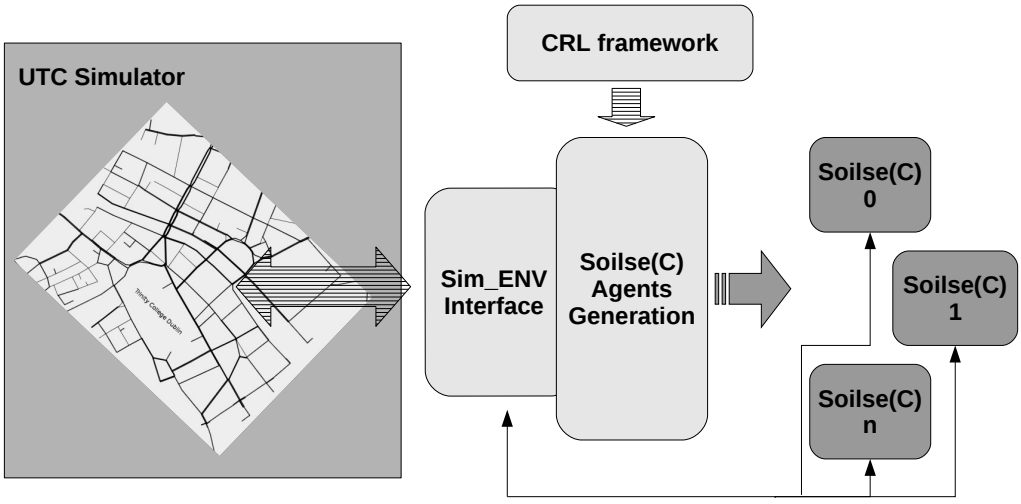


Figure 4.9: High-level agent generation scheme

```
bool monitorPatternChange(double& new_DPC) {

static int steps = 0;
static bool sampling = false;
static double sum_pcd = 0;

double DPC = calcJunctionDPC();

if ( (DPC <= changeThreshold) && !sampling )
        sampling = true;

if ( sampling && (steps < persistence) ) {
        sum_dpc += DPC;
        steps++;

        if (steps == persistence) {
                steps = 0;
        if ( (sum_dpc / persistence) <= changeThreshold) {
                        // New DPC based on DPC sample average
                        new_DPC = (sum_dpc / persistence);
                        sum_dpc = 0;
                        sampling = false;
                        return true; //Persistent change
                }
                sum_dpc = 0;
                sampling = false; //It was a non-persistent change
        }
}
return false;
}
```

Listing 4.5: PCD - monitor pattern change function

Updating the samples needed to calculate the DPC is done through a dedicated function. The $updateVehsCountPerLane()$ function updates the basic sample per lane which consists of vehicle counts. Consequently, the $updateCUSUMsqr()$ updates the input samples needed to calculate the CUSUM of squares using the moving average of basic samples (vehicles counts per lane). The same function updates the CUSUM of squares output sample per lane after the required calculations. Sim-

ilarly, the *updateRewardsSample*() updates the rewards sample. All the update functions mentioned are invoked by the associated Soilse or SoilseC agent through a single function, *updatePCDsamples*(). In order to calculate the final DPC, a dedicated function, *calcJunctionDPC*() calculates both, the mean of all lane moving averages from the $CUSUMsqr\_outputSamplePerLane$ and the moving average of the $Rewards\_Sample$. Furthermore, any sample size must meet the predefined corresponding sample size to carry out any calculation on the sample. The DPC calculation process was detailed in Section (3.3).

4.2.2 Relearn

Upon the *wakeup*() function of a given Soilse or SoilseC agent being called, the function invokes *pcd -> monitorPatternChange()* and the *isLearning*() functions. If the agent was not in a learning status and a new DPC is returned as a result of a persistent pattern change, it calls the *Relearn*() function. This function is characterized by the calculation of new learning and action selection parameters suitable for the detected change and based on the returned DPC value. The latter is passed to the *Relearn*() function which, depending on the learning and action selection strategies types, sets new learning and action selection parameters. The calculation of these parameters was described in Subsection (3.5.1.2).

4.3 Summary

This chapter presented the implementation of our approach in terms of the CRL framework developed to provide the basic constituents needed for building Soilse and SoilseC agents. Indeed, our CRL framework can be used to build other optimization approaches as it is generic in nature. For instance, our CRL framework was extended and used to build a multi-policy optimization scheme for large-scale autonomic systems in (Dusparic & Cahill, 2009a). Both, a Soilse or SoilseC agent-generator customizes and instantiates the CRL framework as well as includes PCD support for all created agents. It also provided an interface that allows for communication with the UTC simulator. The latter is described in the evaluation chapter.

Chapter 5

Evaluation

This chapter details the evaluation of our RL-based approach to optimization of UTC by means of simulation. Our simulations are based on a UTC simulator that was built in the Distributed Systems Group at Trinity College Dublin (TCD). We briefly introduce the possible approaches to UTC simulation and introduce the UTC simulator that we use. We describe the experimental setup in terms of the maps and traffic patterns used, the Soilse and SoilseC learning and action selection settings used as well as the baselines for comparison. The performance metrics we use to evaluate our approach in terms of average vehicle waiting time and average number of vehicle stops are also presented. We experiment with two scenarios of different scale; the first uses a map of TCD and its surroundings and the second, uses a larger map of Dublin inner city centre. We analyse the results obtained from these scenarios and compare them against results from the best-performing baselines. We also evaluate how Soilse and SoilseC scale.

5.1 UTC Simulation

Approaches to traffic simulation have evolved to mainly use one of three models, i.e., microscopic, macroscopic and mesoscopic (Hoogendoorn & Bovy, 2001). The main difference among these models is in the level of granularity at which the traffic dynamics are modelled. We briefly introduce these models:

Microscopic: in this model, the granularity is very fine through assigning every vehicle a specific model of interaction. Usually, a group of vehicles of a given type, e.g., cars, follow the same model. Such a model specifies the vehicle's behaviour in terms of acceleration, deceleration,

car-following, lane-changing and possibly other aspects.

Macroscopic: in this model, the granularity is coarse where traffic flow is modelled based on concepts inspired by fluid dynamics. It deals with vehicles collectively and on homogeneous basis, i.e., does not differentiate between different vehicle types. A group of vehicles are seen as one entity that can be characterized by a given flow-rate, velocity or density.

Mesoscopic: this model bridges the microscopic and macroscopic models. It does not differentiate between different vehicles but rather specifies their behaviour usually probabilistically. Traffic is modelled as small groups of vehicles of given characteristics such as density. Driving related decisions of a given vehicle are relatively affected by the small group of vehicles it belongs to.

The coarse nature of the macroscopic model makes it difficult to represent real-life situations and rather concentrates on providing a less computationally demanding model. On the other hand, microscopic and mesoscopic models seem to be more accurate in describing the low-level dynamics of traffic which makes them more representative of real-life situations. Consequently, we use a microscopic urban traffic simulator.

5.1.1 The UTC Simulator

The UTC simulator (Reynolds et al., 2006) that we use follows a microscopic model. Its input is a set of XML files describing the road network to be simulated and the valid phases for each signalized junction. This includes the number of lanes per road, the maximum allowed speed on a given road, and the distances between connected junctions. Moreover, traffic can be generated between specific junctions or among user-defined zones where the source/destination junctions are selected randomly within the source/destination zones. The resulting traffic data is represented in a trace file fed to the simulator which includes all vehicles insertion times along with each vehicle's respective path, i.e., the sequence of junction identifiers to cross. A snapshot of the UTC simulator's viewer is presented in Figure (5.1).

The traffic data generated by the tools available in the UTC simulator does not provide variability in the traffic trace, i.e., the same input data is produced given the same traffic duration and specific vehicle paths. This can be seen as an evaluation limitation. However, we examine, but not statistically, the effect of varying different parameters such as the ExpFactor and the collaboration frequency where other parameters are automatically computed by the Soilse and SoilseC algorithms. The only uncertainty in Soilse and SoilseC originates from the randomness in the RL action selection strategies used with the exception of pure greedy action selection.

Figure 5.1: UTC simulator - viewer snapshot

In the UTC simulator, vehicles exhibit different behaviours such as car-following, acceleration, deceleration and lane-switching. They also abide by the speed limits on the different roads they travel on. Throughout simulation, the UTC simulator logs vehicles' waiting time and total number of stops and it also provides the throughput at the end represented by the number of vehicles that arrived at their destinations. The UTC simulator was also used in (Dusparic & Cahill, 2009b,a) for evaluating multi-policy optimization schemes in decentralized autonomic systems.

5.2 Experimental Setup

This section specifies the common experimental setup used in the two evaluation scenarios based on the Trinity map and the Dublin inner city centre map. We describe the maps used for each scenario and the traffic patterns used for each. We also present the learning and action selection parameters for Soilse and SoilseC. The baselines we use to compare Soilse and SoilseC against are also presented as well as the metrics used to evaluate the performance.

5.2.1 Maps and Traffic Patterns

Two different maps are used for the two different scenarios. Each scenario also uses a different series of traffic patterns. The traffic patterns used are based on traffic counts deduced from the Dublin

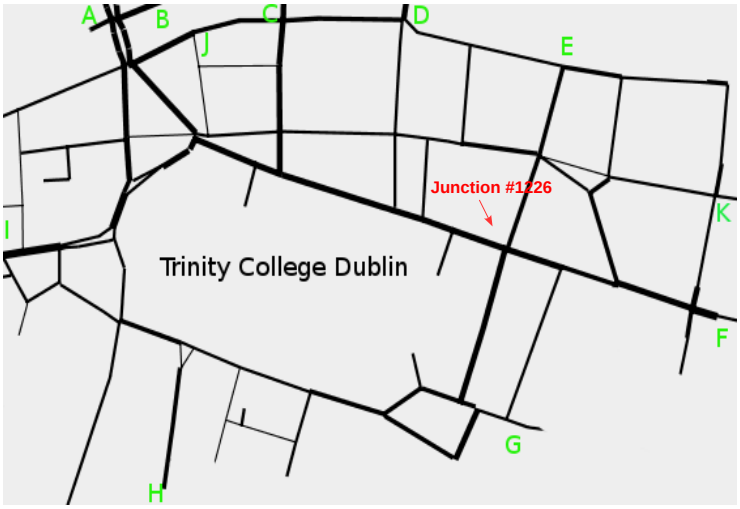


Figure 5.2: Trinity map

From	To	ULP Traffic	MPP Traffic	UHP Traffic	EPP Traffic
F	A	100	2000	1050	x
F	I	100	500	1050	500
H	C	100	500	1050	500
H	G	100	500	1050	500
J	K	100	500	1050	500
B	I	100	500	1050	1500
I	A	100	1500	1050	500
F	C	100	500	1050	500
E	G	100	1500	1050	x
D	A	100	500	1050	500
H	A	x	500	x	500
E	A	x	500	x	500
E	I	x	500	x	500
B	F	x	x	x	2000
H	E	x	x	x	1500

Table 5.1: Trinity scenario traffic patterns

Transportation Office Road Users Monitoring Report (Dublin Transportation Office, 2008).

Trinity scenario: this scenario uses a map (see Figure (5.2)) that represents the real road network of TCD and its surroundings (Trinity map). It consists of 104 junctions, 30 of which are signalized junctions that need to be controlled. The overall traffic duration is $\sim$ 19 hours comprising four different patterns:

Uniform low pattern (ULP): traffic is generated over $\sim$ 4 hours following a uniform pattern

of low traffic load consisting of 1000 vehicles. It is uniform in the sense that traffic is generated in equal proportions from all possible sources (see the From column in Table (5.1)) to all possible exits (see the To column in Table (5.1)). Cases where a given source is also an exit are avoided. See Table (5.1).

Morning peak pattern (MPP): traffic is generated over ~ 4 hours following a pattern that reflects high traffic loads on main roads comprising $10,000$ vehicles. See Table (5.1).

Uniform high pattern (UHP): traffic is generated over ~ 7 hours following a uniform pattern of high traffic traffic load consisting of $10,500$ vehicles. See Table (5.1).

Evening peak pattern (EPP): traffic is generated over ~ 4 hours following a pattern that reflects high traffic loads on main roads consisting of $10,000$ vehicles but generally opposing in direction to the morning peak pattern. See Table (5.1).

Dublin inner city centre scenario: this scenario uses a map (see Figure (5.3)) that represents the real road network of a considerable portion of Dublin city centre. In this scenario, 62 signalized junctions need to be controlled out of the overall total of 270 junctions. This size is comparable to a city centre, $\sim 62.6\%$ of the size of Cork city centre, which has 99 signalized junctions controlled by SCOOT (personal communication). The overall traffic duration is ~ 19 hours comprising four different patterns:

ULP: traffic is generated over ~ 4 hours following a uniform pattern of low traffic load consisting of 2000 vehicles. Indeed, cases where a given source is also an exit are avoided.

MPP: traffic is generated over ~ 4 hours following a pattern that reflects high traffic loads on main roads consisting of $20,000$ vehicles.

UHP: traffic is generated over ~ 7 hours following a uniform pattern of low traffic load consisting of $21,000$ vehicles.

EPP: traffic generated over ~ 4 hours following a pattern that reflects high traffic loads on main roads consisting of $20,000$ vehicles but generally opposite in direction to the morning peak pattern.

Traffic patterns used for the Dublin inner city centre scenario are analogous to those used for the Trinity scenario. However, much higher traffic volumes are used and parking space areas are introduced (see Figure (5.3)) in order to be used for generating morning and evening peak patterns. These parking space areas are chosen in realistic locations around Dublin inner city centre. For the ULP and UHP traffic patterns for the Dublin inner city centre scenario, traffic is generated in both

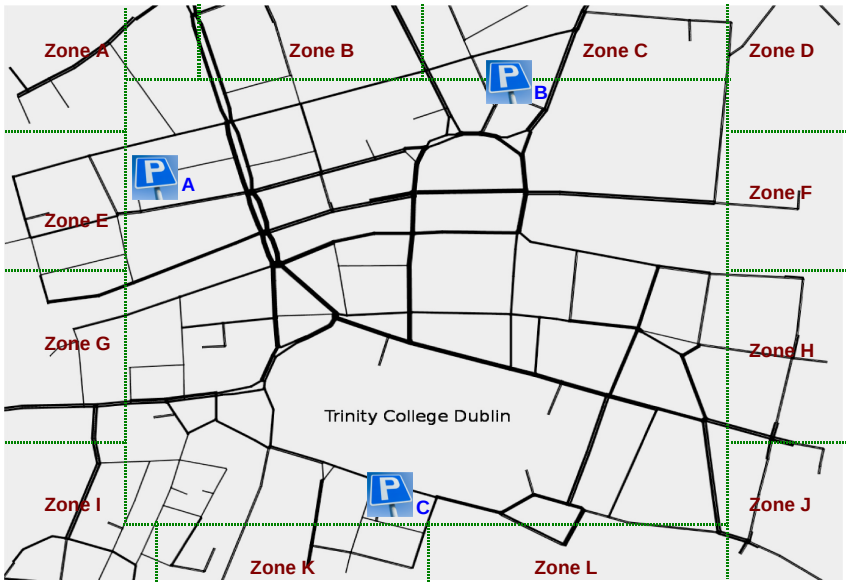


Figure 5.3: Dublin inner city centre map

directions from all opposing edges of the map uniformly. To clarify, traffic incoming from different sources in *zones* $\{A, B, C, D\}$ is destined for different exits on *zones* $\{I, K, L, J\}$ and vice versa. Also, traffic incoming from different sources in *zones* $\{A, E, G, I\}$ is destined for different exits on *zones* $\{D, F, H, J\}$ and vice versa. The difference between ULP and UHP is only in the traffic load and in the pattern duration. For the MPP traffic pattern in Dublin inner city centre scenario, two types of traffic are generated, the first is heavy traffic destined for the different parking spaces and the second is light uniform traffic similar to the ULP. Incoming heavy traffic in that case arrives to a given parking space from all remote zones, for example, *parking space* A would receive heavy traffic from all zones except near *zone* E (as vehicles will arrive almost immediately) and so on. Such heavy traffic amounts to nearly 6324 vehicles per parking space area over the MPP duration. The opposite happens in the EPP where heavy traffic leaves form the parking space areas to all remote zones. Traffic loads leaving the parking spaces are similar in load to those in the MPP. As well, light uniform traffic similar to the ULP is generated simultaneously.

Both scenarios presented are simulated with the combined set of their respective patterns. To clarify, the Trinity scenario would run for the joint series of its patterns, i.e., $ULP \rightarrow MPP \rightarrow UHP \rightarrow EPP$ for a duration slightly more than 19 hours to allow for the most recent traffic to clear the map. The same is the case for Dublin inner city centre scenario using its respective patterns. Specifically, ULP starts the simulation, MPP follows at $\sim 14400sec$, then UHP follows at $\sim 2900sec$ and finally EPP follows at $\sim 54000sec$ in the simulation.

5.2.2 Soilse and SoilseC Specifics

Pivotal constituents of the Soilse and SoilseC agents are the learning and action selection strategies as well as the Pattern Change Detection (PCD) mechanism. In our scenarios, Soilse and SoilseC agents share the following common learning, action selection, and PCD specifics (listed also in Table (5.2)):

Learning strategy: Q-learning is used as the learning strategy with α (learning rate) initially set to a high 0.99 and gradually decreasing based on an initial $\alpha_{decay\,rate} = 0.03$. This allows α to reach the minimum value of 0.001 after $\sim$ 115 minutes. Two discount factors (γ) are fixed to either 0.3 or 0.7 throughout the scenario simulation which are representative of mid-low and mid-high γ values respectively.

Action selection strategy: three strategies can be used; $\epsilon-$greedy, greedy or Boltzmann. Concerning $\epsilon-$greedy, its initial ϵ value is set to a high 0.99 and gradually decreases based on an initial $\epsilon_{decay\,rate} = 0.03$. Analogous to Q-learning's α, ϵ reaches the minimum value of 0.001 after $\sim$ 115 minutes. In case of Boltzmann, the initial temperature τ is set to 1000 and it gradually cools down to the minimum value of 1 based on a temperature decay rate $\tau_{decay\,rate} = 0.03$ after $\sim$ 115 minutes. In case of using greedy action selection strategy, only Q-learning's α is affected.

PCD: Concerning the sample size needed for the CUSUM of squares on a lane, $n = 30$ and $k = 15$ are used. The moving average filter on each lane's traffic has a moving sample size of 60 traffic counts collected every second. Concerning the smoothing moving average filter on the CUSUM of squares output, a sample size of 10 is used. Also, the reward moving average filter uses a moving sample of 10 rewards. A fixed thresholding technique is used with a junction change threshold set to 0.85. A *PerFactor* of 10 is used to squash the resulting Degree of Pattern Change (DPC) while the persistence sample has a size of 10. See Section (3.3.2) for details on how these values are selected.

In both Soilse and SoilseC, relearning is evaluated with three values of *ExpFactor* (exploration factor, see Section (3.5.1.2)) $\in \{1, 2, 5\}$. The latter set of *ExpFactor* values was chosen as agents would spend a long time when relearning if the *ExpFactor* exceeds the value 5, which consequently makes them unresponsive to possible genuine changes in traffic. Given that the *ExpFactor* value affects the relearning period, a set of values (≤ 5) were calculated to assess its effect on the overall performance. The threshold used to determine a phase's busy status is set to 1. The action set used is composed of 0, 20 and 30 second available for each state in the Soilse or SoilseC state-action space. A Soilse agent uses R1 (see Equation (5.1), refer to Section (3.5.1.1) for details) while a SoilseC agent uses

Parameter	Value(s)
Initial α, ϵ	0.99
Initial $\alpha_{decay\,rate}$, $\epsilon_{decay\,rate}$, $\tau_{decay\,rate}$	0.03
γ	$\{0.3, 0.7\}$
Initial τ	1000
CUSUM k, n	30, 15
CUSUM (input, output) sample sizes	(60, 10)
Junction change threshold	0.85
PerFactor	10
Persistence sample size	10
Rewards history length	10
ExpFactor	{1, 2, 5}
Durations of the three actions used	$(0, 20, 30)sec$
Phase threshold	1
NPV discount rate	0.1
Collaboration frequency	$\{120, 240\}sec$

Table 5.2: Experimental parameters

a collaborative reward model that incorporates R1 as local reward model (see Equation (3.10) and Section (3.5.2.2) for details).

$$R1 = (\,number\,of\,vehicles_{\,crossed} - number\,of\,vehicles_{\,waiting\,on\,the\,junction}\,) \tag{5.1}$$

Every SoilseC agent follows an advertisement strategy of Net Profit Value (NPV) discount rate 0.1 while we experiment with two collaboration frequencies of 120 or 240 seconds (see Section (3.5.2) for details on how these values are selected). We also evaluate the three modes of collaboration forming the neighbours to send to and those to receive rewards from as described in Section (3.5.2.1). The type of phases each signalized junction uses are concise phases and are described in Section (3.4). Finally, all agents' initial learning occurs on the ULP where this learning lasts for ~ 115 minutes.

5.2.3 Baselines for Comparison

We use two baselines for comparison to compare Soilse and SoilseC performance against:

Round robin (RR): this baseline simply allows every signalized junction controller to cycle through the available phases while giving an equal amount of time to each. We use 20, 30 and 40 seconds in our scenarios. For example, $RR20s$ running on a signalized junction of 3 phases would have a cycle time of 3×20 seconds. This means that the time given to each phase is fixed and hence RR represents a fixed-time UTC plan.

SAT: an algorithm (Richter, 2006) that emulates the behaviour of SCATS by trying to achieve a 90% saturation level at signalized junctions. The saturation level in this sense depends on the efficiency of using the available green time. We set the minimum phase time to 20 seconds and the SAT controllers determine the maximum cycle length based on $[minimum\,phase\,time \times max\_cycleL\_factor \times number\,of\,phases]$. The max_cycleL_factor we choose for our scenarios are 1.1 and 1.5 and the number of phases depends on the junction being controlled. The SAT controllers try to adapt according to the saturation level by incrementing or decrementing phase durations at the beginning of each cycle depending on information from the previous cycle. The decrement or increment amount is a fixed number (DIM). We choose to experiment with 2 and 5 seconds in that case which were found to provide better performance in SAT compared to other tried values. Henceforth, a SAT controller is described as in $SAT\_\{DIM\}\_\{max\_cycleL\_factor\}$, e.g., SAT_2_1.5.

The above baselines are representative of two UTC schemes, namely, RR's fixed-time control and SAT's adaptive control. Both present reasonable competitiveness depending on the scenario on which they are deployed as the results discussed further on will show.

5.2.4 Performance Metrics

We rely on two metrics to assess the performance of Soilse and SoilseC against the baselines in the two scenarios described before.

Waiting time: for a given vehicle, the waiting time (also known as delay) represents the amount of time that the vehicle is motionless throughout the journey to a given destination. Intuitively, UTC optimization aims at reducing that time.

Number of vehicle stops: for a given vehicle, the number of stops represents the instances at which its velocity reached *zero* throughout the journey to a given destination. Intuitively, UTC optimization aims at reducing the number of stops per vehicle.

As we are dealing with traffic consisting of many vehicles, we need a collective metric. Hence, the average waiting time (AWT) and the average number of vehicle stops (AvgStops) per arrived vehicle are used as collective metrics. As metrics for the ongoing performance, we monitor the (accumulated) total number of stopped vehicles and the (accumulated) total waiting time of all vehicles present at a given time throughout a given simulation scenario. The number of arrived vehicles is also presented which represents the overall throughput.

Better performance in terms of the aforementioned metrics is crucial to UTC performance. They convey the degree to which the ultimate goal of decreasing congestion is met. They are also common metrics in the UTC optimization literature. As argued in (Klein, 2001), vehicle delay or waiting time and the number of stops vehicles suffer are important measures of traffic congestion and traffic flow. Particularly, they are important for studying the environmental effect of traffic such as fuel consumption and emissions.

We use, Soilse or SoilseC to refer to a deployment where every signalized junction in a given scenario is controlled by a Soilse or a SoilseC agent respectively, unless there is an explicit use of Soilse agent(s) or SoilseC agent(s). Performance results are presented with the prefix ($\sim -$) or ($\sim +$) to indicate an approximate lower or higher value respectively, for example, if a given Soilse deployment outperforms the RR20s baseline by $\sim -x\%$ in terms of AWT, this implies that Soilse provides $\sim x\%$ lower AWT than RR20s.

Plots and graphs showing AWT, AvgStops and the ongoing performance based on total waiting time and total number of stops do not include a measure of uncertainty such as standard deviation. The reason is due to a limitation in the UTC simulator (see Section (5.1.1)) which does not support the variability of input data for same traffic patterns that can be used for different runs.

5.2.5 Evaluation Objectives

The evaluation that we carried out addresses whether our approach meets the following objectives.

Objective one (Obj1): assess the benefits of relearning in our approach against a situation where initial learning is only used. In addition to assessing the performance of our approach using the available action selection strategies.

Objective two (Obj2): assess the relearning behaviour in our approach.

Objective three (Obj3): assess the viability of our approach for UTC by comparing Soilse and SoilseC performance against the baselines performance.

Objective four (Obj4): assess how collaboration using SoilseC can provide better global performance against non-collaborative Soilse. In addition to assessing available collaboration modes.

Objective five (Obj5): assess how our approach scales in terms of map size and traffic loads.

We refer to the above objectives individually as we address each in the evaluation of our approach using two scenarios of different scales. In addition, the same objectives are addressed in both scenarios.

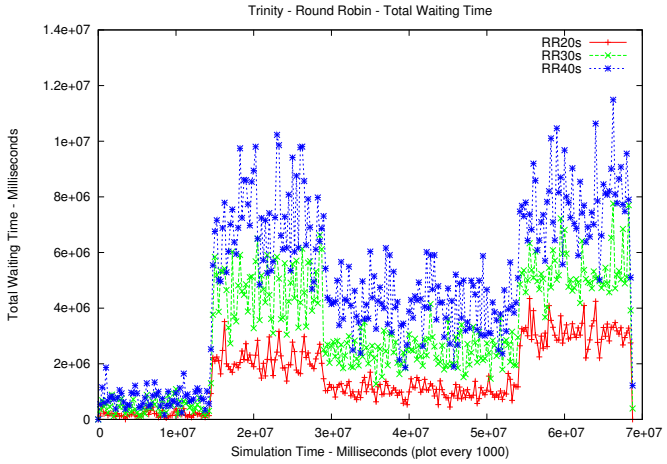


Figure 5.4: Trinity - RR total vehicle waiting time throughout the simulation time

5.3 Trinity Scenario

The map used for this scenario and the series of combined patterns were described in Section (5.2.1). First we introduce the results from the baselines evaluation and then compare their performance in order to obtain the best performing baseline settings. The latter's performance are used for further comparison against the performance of Soilse and SoilseC agents. Also, Soilse is compared against a situation where an initial learning is only used. For this scenario, we also provide different levels of comparison between different parameters of Soilse and SoilseC. Specifically, we study the effect of using different exploration factors on the relearning behaviour. The latter is also discussed for Soilse and SoilseC. Furthermore, we study the effect of the collaboration mode in use as well as the frequency at which SoilseC agents collaborate. Finally, we compare the results of the best performing baseline settings against the best Soilse and SoilseC performance.

5.3.1 Baseline Performance

We present the performance of the different baselines as a basis for selecting the best performing baselines for further comparison against Soilse and SoilseC.

Figures (5.4) and (5.5) show the total waiting time for all vehicles in the simulation at a given time for all RR and SAT settings respectively. It is evident that RR20s outperforms RR30s and RR40s in terms of total vehicle waiting time throughout the simulation time. Given the inflexibility of RR in terms of changing phase timings or avoiding certain unnecessary phases for a given traffic pattern, it appears that for high phase times such as 30 or 40 seconds, a considerable portion of the phase

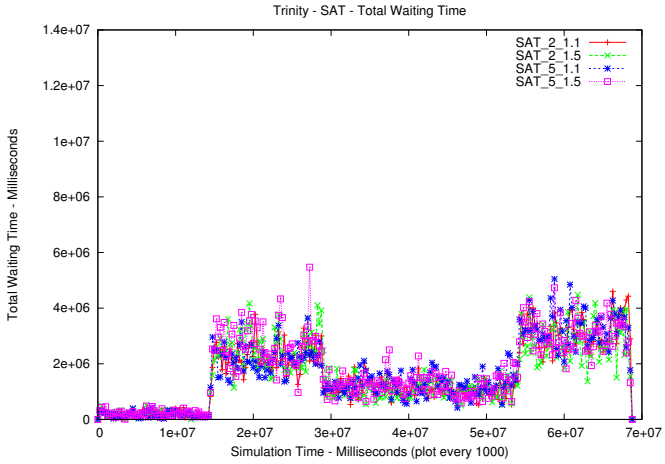


Figure 5.5: Trinity - SAT total vehicle waiting time throughout the simulation time

time is wasted. This causes vehicle queues to build up on certain busy lanes as they are obliged to wait for their turn (phase) in order to be allowed to cross after the previous phase time elapses. If a lane is served by a given phase at a signalized junction with 3 phases that runs RR40s, it would need to wait 2×40 seconds for its turn to be served. Given that RR20s allows for cycling phases to end relatively faster, it allows vehicles on more approaches to cross more frequently resulting in better performance than RR30s and RR40s in this scenario. As far as SAT is concerned, different SAT settings, namely, SAT_2_1.1, SAT_2_1.5, SAT_5_1.1 and SAT_5_1.5 perform approximately on a par in terms of total vehicle waiting time throughout the simulation time. In comparison to RR performance, Figure (5.6) shows the AWT results for different RR and SAT settings. In terms of AWT, RR20s ($\sim 34s$) slightly outperforms the best performing SAT, i.e., SAT_2_1.5 ($\sim 37.88s$). RR30s and RR40s perform the worst in terms of AWT. Concerning the performance in terms of AvgStops (see Figure (5.7)), most of SAT experiments perform nearly on par with the best performing RR20s ($AvgStops = \sim 4.78$) with the exception of SAT_2_1.1.

As far as the number of vehicles that arrived over the simulation duration is concerned, Figure (5.8) shows the performance of all baselines with their different settings. SAT_2_1.5 performs best by allowing $30,162$ vehicles to arrive as opposed to the best performing RR, i.e., RR20s that allowed only $29,128$ vehicles to do so. This represents $\sim 3.2\%$ better performance is terms of the number of arrived vehicles.

Based on these results, RR20s and SAT_2_1.5 represent the best performing baselines. The difference between the selected baselines for this scenario in terms of total vehicle waiting time throughout the simulation time is presented in Figure (5.9). It shows that both selected baselines perform simi-

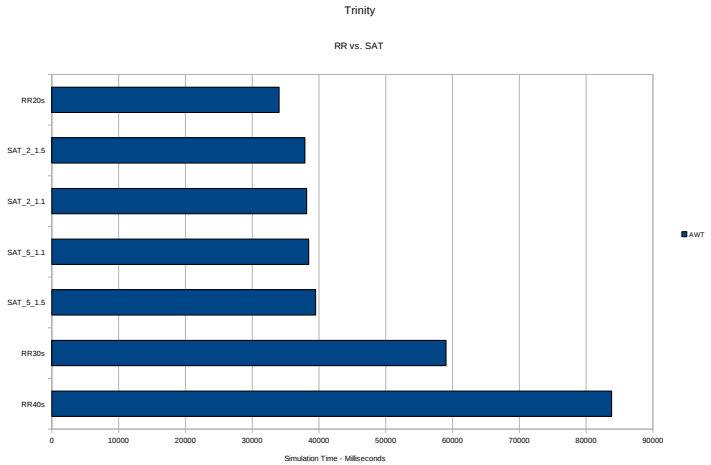


Figure 5.6: Trinity - RR (20s, 30s, 40s) vs. SAT (2_1.1, 2_1.5, 5_1.1, 5_1.5) - AWT

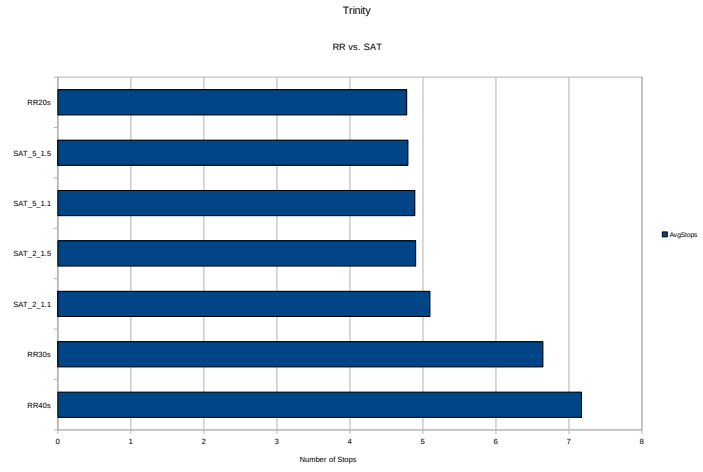


Figure 5.7: Trinity - RR (20s, 30s, 40s) vs. SAT (2_1.1, 2_1.5, 5_1.1, 5_1.5) - AvgStops

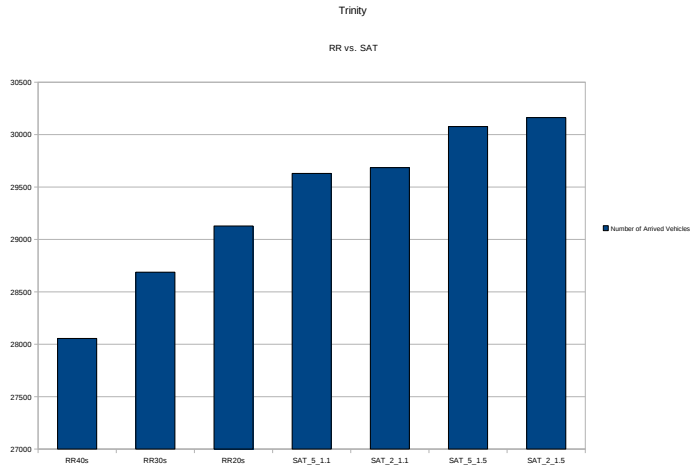


Figure 5.8: Trinity - RR (20s, 30s, 40s) vs. SAT (2_1.1, 2_1.5, 5_1.1, 5_1.5) - number of arrived vehicles

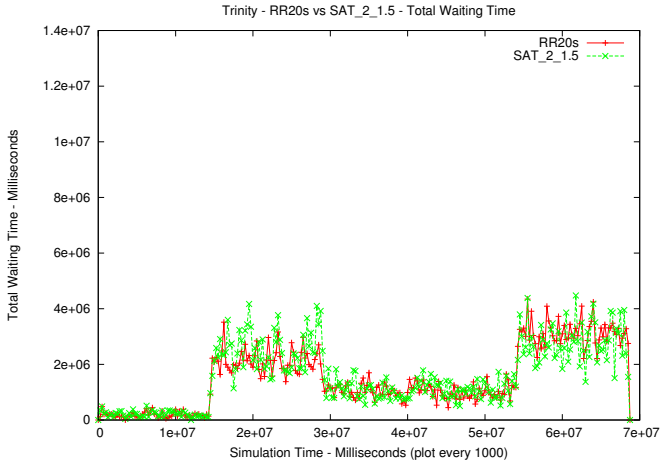


Figure 5.9: Trinity - RR20s vs SAT_2_1.5 - total vehicle waiting time throughout the simulation time

larly with respect to the total vehicle waiting time but with slightly lower performance by SAT_2_1.5 under the MPP (see Figure (5.9)). However, RR20s accumulates less vehicle waiting time after the ULP fades away (see Figure (5.10)). Figure (5.11) presents the selected baselines' performance in terms of the total number of stopped vehicles at a given time during the simulation. The difference is not clear, however, SAT_2_1.5 exhibited a higher number of stopped vehicles towards the end of the simulation.

The selected baselines are compared against all others in terms of AWT as a representative metric in Table (5.3) which presents a percentage comparison in terms of AWT between the selected baselines

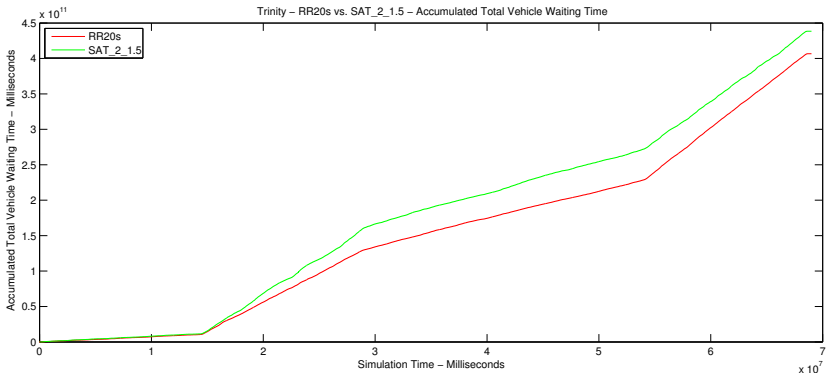


Figure 5.10: Trinity - RR20s vs SAT_2_1.5 - accumulated total vehicle waiting time

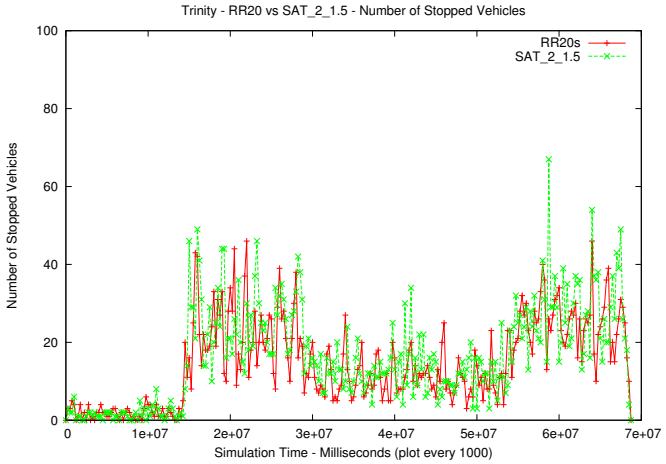


Figure 5.11: Trinity - RR20s vs. SAT_2_1.5 - number of stopped vehicles throughout the simulation time

AWT %*	RR20s	SAT_2_1.5
RR 30s	-42%	-36%
RR 40s	-59%	-55%
SAT_2_1.1	-11%	-1%
SAT_5_1.1	-12%	-2%
SAT_5_1.5	-14%	-4%
SAT_2_1.5	-10%	x

*The negative sign (-) implies lower AWT

Table 5.3: Trinity - selected baselines AWT performance comparison - Trinity map

and all others.

It is clear that in terms of AWT, RR20s outperforms all SAT experiments while the best performing SAT_2_1.5 outperforms all other SAT as well as RR30s and RR40s.

5.3.2 Soilse

In this section we evaluate the performance results of Soilse deployment in the Trinity scenario where a dedicated Soilse agent is assigned to control every signalized junction. We analyse the effect of different action selection strategies in Soilse. Moreover, we analyse the effect of the *ExpFactor* on the relearning. In addition to comparing the performance of Soilse against the performance of the selected baselines, we also compare the performance of Soilse against the performance of SoilseInit that is a situation where initial learning without relearning is only used. This is done in order to show the validity of relearning as a means to provide for responsiveness and adaptiveness in Soilse. It is worth mentioning that all metric results for AWT and AvgStops provided for Soilse are based on the simulation period only after the initial learning has finished and on the averaged results of three runs. The variability in these runs originates solely from the random functions used in the deployed action selection strategy apart from pure greedy (see Section (5.1.1)). Plots of (accumulated) total vehicle waiting time and number of stopped vehicles throughout the simulation period are based on the best run from the three runs.

5.3.2.1 Initial Learning vs. Relearning

We first compare the performance of Soilse using different action selection strategies against the corresponding SoilseInit (see Obj1). Table (5.5) shows the best performance of Soilse in terms of AWT for each action selection strategy against SoilseInit. We first discuss the results from Soilse using different action selection strategies.

	SoilseInit			Soilse			Soilse Performance$^{\mp}$		
	∼AWT*	#AV	∼AvgStops	∼AWT*	#AV	∼AvgStops	∼AWT%	#AV%	∼AvgStops%
ϵ-greedy	35.606	29422	4.38	30.484	30152	4.37	-14.38%	+2.42%	-0.228%
Greedy	36.603	29540	4.47	33.345	29651	4.47	-8.90%	+0.37%	0%
Boltzmann	47.812	29297	5.97	44.864	29448	5.74	-6.16%	+0.51%	-3.69%

*Numbers are in seconds.

Table 5.4: Trinity - Soilse vs. SoilseInit based on best AWT performance per action selection strategy

Soilse Performance$^{\mp}$	ϵ-greedy		
	∼AWT%	∼AvgStops%	∼#AV%
Greedy	-8.57%	-2.23%	+1.66%
Boltzmann	-32.05%	-23.86%	+2.33%

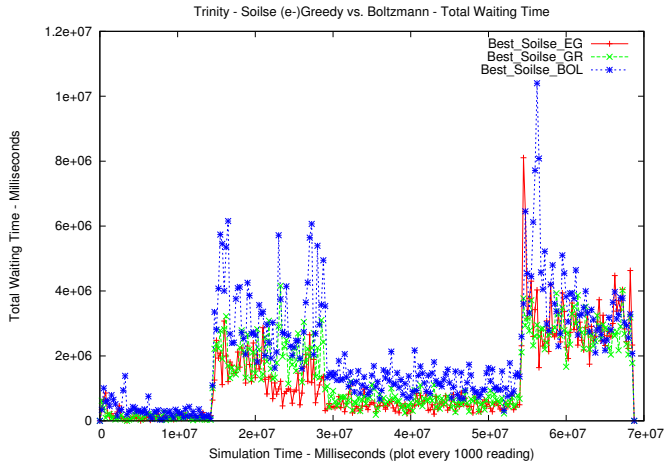
$^{\mp}$The negative sign (-) indicates lower AWT, AvgStops or #Arrived Vehicles (AV), otherwise (+) is used.

Table 5.5: Trinity - best AWT Soilse performance per action selection strategy

From the results presented in Table (5.5), it transpires that Soilse using ϵ-greedy action selection strategy outperforms Soilse using Boltzmann and Soilse using greedy by $\sim -32.05\%$ and $\sim -8.57\%$ respectively in terms of AWT. Furthermore, results in terms of AvgStops show that using ϵ-greedy in Soilse also results in better performance compared to using Boltzmann and greedy by $\sim -23.86\%$ and $\sim -2.23\%$ respectively. Concerning the number of vehicles that arrived at their destinations, Soilse using ϵ-greedy has marginally outperformed both Soilse using Boltzmann and Soilse using greedy by allowing $\sim +2.33\%$ and $\sim +1.66\%$ more vehicles to arrive respectively.

Furthermore, addressing Obj1 in terms of relearning benefits, results based on best AWT performance for Soilse against SoilseInit presented in Table (5.4) show that in all the cases Soilse, by relearning when a genuine traffic pattern change occurs, performed better especially in terms of AWT against SoilseInit. In terms of AWT, Soilse outperformed SoilseInit by providing $\sim -14.38\%$, $\sim -8.90\%$ and $\sim -6.16\%$ better performance in cases where ϵ-greedy, greedy and Boltzmann were used respectively.

Figures (5.12)(5.13) reaffirm the results concerning the different action selection strategies used in Soilse in terms of AWT. Using Boltzmann in Soilse resulted in poor vehicle waiting time performance in this scenarios as opposed to using ϵ-greedy or greedy. This is clear in the previous figures as Soilse using Boltzmann does not manage to learn or relearn better policies compared to Soilse using ϵ-greedy and greedy. It is also noticeable in Figure (5.12) that surges in the total waiting time occur as some Soilse agents relearn. This occurs in all action selection strategies, on different signalized junctions and at different levels of relearning, (i.e., due to different reparameterization). Monitoring the accumulation of



*EG: ϵ-greedy - GR: Greedy - Boltzmann: BOL

Figure 5.12: Trinity - Soilse using Boltzmann vs. (ϵ-)greedy - total waiting time throughout the simulation time

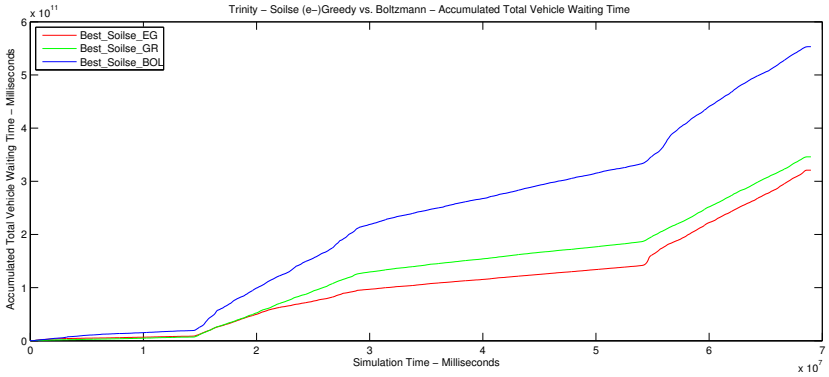


Figure 5.13: Trinity - Soilse using Boltzmann vs. (ϵ-)greedy - accumulated total waiting time throughout the simulation time

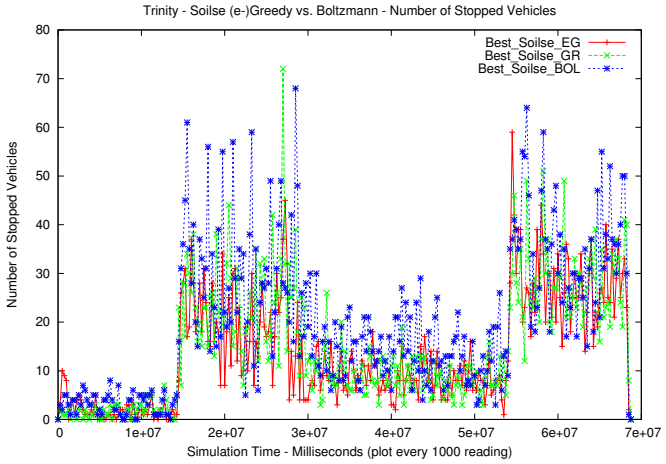


Figure 5.14: Trinity - Soilse using Boltzmann vs. (ϵ-)greedy - number of stopped vehicles throughout the simulation time

total vehicle waiting time (see Figure (5.13)) clarifies how Soilse using Boltzmann performance cannot cope adequately in this scenario as opposed to its ϵ-greedy and greedy counterparts. Furthermore, Soilse's performance in terms of the number of stopped vehicles is presented in Figure (5.14). It is clear that Soilse using Boltzmann caused a higher number of stopped vehicles over most of the simulation as opposed to Soilse using ϵ-greedy and greedy. This performance is reminiscent of the total waiting time performance throughout the simulation time.

In summary, Obj1 is satisfied in this scenario as we showed how Soilse, by relearning, outperforms SoilseInit in all action selection strategy cases. Better performance was achieved in Soilse using ϵ-greedy against its SoilseInit counterpart in terms of AWT, i.e., $\sim -14.38\%$. In addition, an assessment for Soilse using different action selection strategies is provided which also satisfies Obj1.

As far as Soilse's action selection strategy is concerned, ϵ-greedy's overall performance in this scenario is better than other action selection strategies in terms of AWT and AvgStops. We believe this is due to the efficient manner in which ϵ-greedy relearns. This normally occurs with minimal relearning cost, (i.e., in terms of total vehicles time and number of stopped vehicles) that does not leave a severe negative effect on the overall system in addition to reaching better policies after each relearning. The nature of ϵ-greedy's exploration does not rely heavily on the underlying policy, which is a good feature when trying to relearn a different policy for some emerging traffic pattern. On the contrary, Boltzmann's exploration process is affected by the current policy which makes it harder to relearn a different one in a short period of time when a new traffic pattern emerges. Hence, Soilse using Boltzmann cannot reach better policies after relearning that can be considered better than ϵ-

	ExpFactor 1		ExpFactor 2		ExpFactor 5	
	∼AWT*	∼AvgStops	∼AWT*	∼AvgStops	∼AWT*	∼AvgStops
ϵ-greedy	32.364	4.33	31.765	4.17	30.484	4.37
Greedy	35.151	4.67	33.345	4.47	33.963	4.58
Boltzmann	45.401	5.63	45.204	5.76	44.864	5.74

*Numbers are in seconds

Table 5.6: Trinity - best AWT Soilse performance per exploration factor (ExpFactor)

greedy. Furthermore, Soilse using greedy performs better than using Boltzmann in terms of AWT and AvgStops but does not outperform Soilse using ϵ-greedy. This is due to the exploration control parameter(s) which in the case of Soilse using greedy is only α while in ϵ-greedy, ϵ which determines the randomness of action selection has shown to be beneficial. Upon the detection of a traffic pattern change, Soilse using greedy would only relearn based on the sole greedy action selected. The latter's reward value is used to update the policy only to the extent defined by the newly calculated but decaying α. On the other hand, ϵ-greedy allows for non-greedy but randomly (to a certain ϵ degree) selected actions to be explored, which often results in better overall performance in the near future during the emerging traffic pattern than selecting pure greedy actions during exploration.

5.3.2.2 Relearning Behaviour

Soilse's relearning behaviour is analysed here in order to address Obj2. The effect of the exploration factor value on relearning as well as relearning occurrences and the ratio of relearning time to the simulation time are discussed. Table (5.6) presents the best performing Soilse results in terms of AWT per *ExpFactor* and action selection strategy. It also includes the corresponding AvgStops results.

Different *ExpFactor* values affect Soilse's performance in terms of AWT and AvgStops under all action selection strategies to different extents. Soilse using greedy under *ExpFactor* 2 performed $\sim -2.41\%$ and $\sim -4.28\%$ moderately better in terms of AvgStops than under *ExpFactor* 5 and *ExpFactor* 1 respectively. The suitability of *ExpFactor* 2 in that case is due to a relearning behaviour that balanced between the relearning cost and overall performance. Consequently, Soilse using greedy under *ExpFactor* 2 provided $\sim -1.81\%$ and $\sim -5.13\%$ lower AWT than under *ExpFactor* 5 and *ExpFactor* 1 respectively. On the other hand, Soilse using ϵ-greedy and Boltzmann showed a steady improvement in AWT performance as the *ExpFactor* value increased. For example, a $\sim -5.80\%$ lower AWT resulted from Soilse using ϵ-greedy at *ExpFactor* 5 as opposed to the case at *ExpFactor* 1. It was noticeable that an improvement in AWT was not always accompanied by an improvement in AvgStops. Soilse using ϵ-greedy and Boltzmann performed best at *ExpFactor* 2 and *ExpFactor* 1

respectively in terms of AvgStops. However, Soilse using greedy performed best at *ExpFactor* 2 in terms of both AWT and AvgStops. It transpires that there is no guarantee of obtaining simultaneous best performance in terms of AWT and AvgStops under all *ExpFactor* values for Soilse in this scenario. This may be due to the nature in which relearning occurs at different signalized junctions especially with different DPCs and no collaboration.

We selected signalized junction #1226 (Pearse street and Lombard/Westland Row street crossing) to show the relearning periods for Soilse using ϵ-greedy under different *ExpFactor* values. Figure (5.15) presents the progress of local traffic pattern change on that signalized junction depicted as varying DPC values, which also incorporate the local Soilse agent's performance. As a response to genuine persistent changes in the DPC value, relearning periods are initiated as seen in Figure (5.16) that represents the changes in ϵ of the Soilse agent at junction #1226. Q-learning's α uses the same initial value and decay rate as ϵ during relearning and hence is not plotted.

Based on Figures (5.16) and (5.15), it is clear that the Soilse agent on junction #1226 detected the emergence of all patterns under all *ExpFactor* values except for *ExpFactor* 5 as it had a longer relearning period for the MPP, which caused it to miss the emergence of UHP. Naturally, the new ϵ value decayed at different rates depending on the *ExpFactor* value and the DPC of every change. Moreover, using *ExpFactor* 1 caused the Soilse agent to relearn twice more after the initial relearning was completed for the MPP and the start of the relearning period caused by the emergence of UHP. This is due to the agent's poor performance after the initial relearning period given *ExpFactor* 1, which caused a further need to relearn. This was caused by persistent low DPC values. The situation under *ExpFactor* 2 appears to be good since relearning occurred as the Soilse agent responded at each traffic pattern emergence, however, this might not have been a good situation for other agents. The use of *ExpFactor* 2 for this specific Soilse agent resulted in relatively stable high DPC values during the MPP. Under *ExpFactor* 5, the Soilse agent relearnt for longer periods per genuine traffic pattern change. The first relearning task lasted long enough to miss the chance to relearn for the next emerging pattern UHP. However, the policy that was reached, which might have been affected by parts of the emerging UHP on a low exploratory rate, did not severely affect the local Soilse agent performance under UHP and hence there was no further relearning until a new genuine change was detected due to the emergence of the EPP.

Furthermore, certain signalized junctions (each controlled by a Soilse agent) have relearnt during the simulation time. Figure (5.17) shows the start and end times of relearning periods for all Soilse agents including the initial learning period (please note from now on that junction ID 50 corresponds to junction#1226 presented earlier). It can be observed that all Soilse agents have started and ended

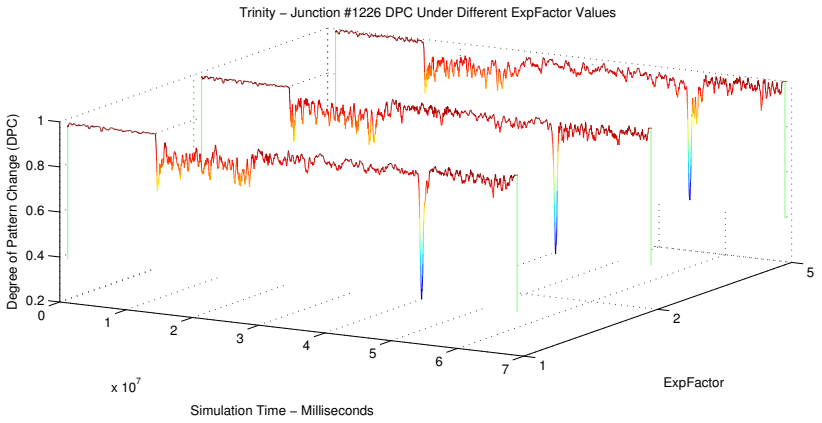


Figure 5.15: Trinity - junction #1226 DPC under Soilse using ϵ-greedy for different ExpFactor values

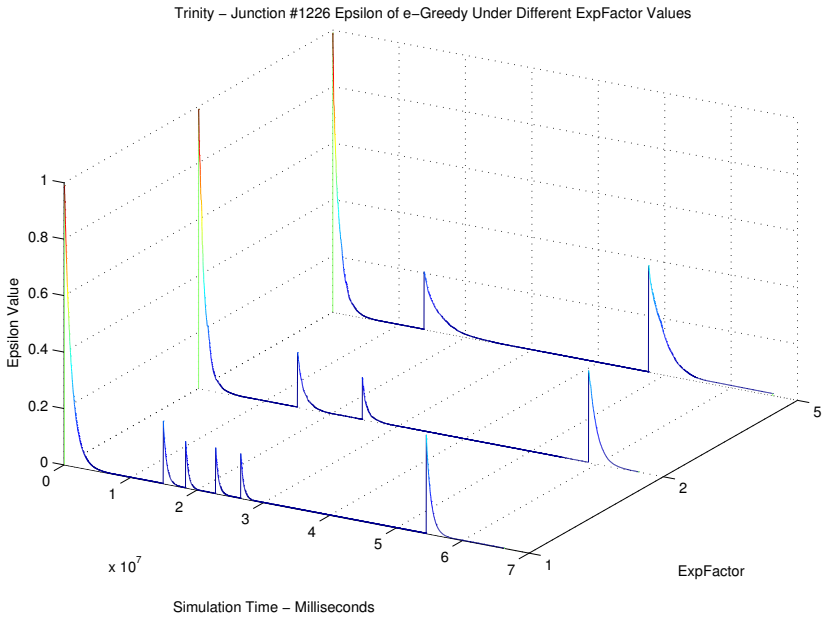


Figure 5.16: Trinity - junction #1226 ϵ-greedy epsilon change using Soilse under different ExpFactor values

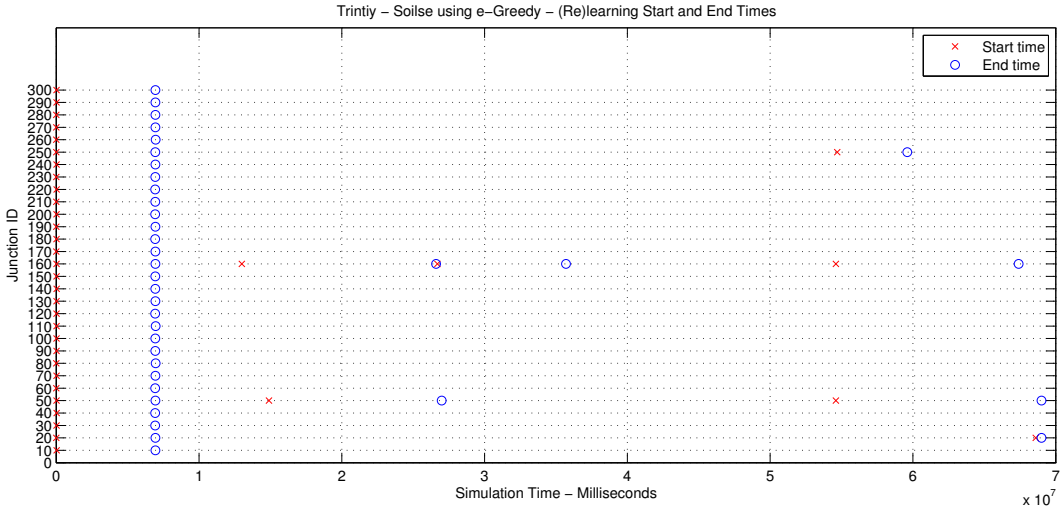


Figure 5.17: Trinity - Soilse using ϵ-greedy (re)learning start and end times - best AWT performance

the initial learning at the same time. In total, four Soilse agents have relearnt at different times. Three Soilse agents have experienced major relearning periods while another single Soilse agent (junction ID 20) have experienced little relearning. It can be noticed that not all agents start relearning at a given time and this is due to a combination of sensitivity and the varying degree of pattern change at the controlled junctions. Bearing in mind that the *ExpFactor* of this best Soilse performance using ϵ-greedy was 5, the relearning periods especially at junctions IDs 50 and 160 were lengthy taking into account the degree of genuine traffic change at each as well. Soilse agent at junction ID 50 was clearly affected by the MPP and EPP while that on junction ID 250 only was affected by the EPP. However, Soilse at junction ID 160 was affected the most by the MPP, the UHP and the EPP. Figure (5.18) shows the time spent in relearning in comparison to the simulation duration for the affected Soilse agents. Soilse agents at junction ID 50 and ID 160 spent $\sim 38\%$ and $\sim 43\%$ of the simulation time respectively in relearning. However, during relearning, the exploration is gradually decreased towards more exploitation so the ratio of relearning time to the simulation time indeed includes a continuously increasing exploitation.

In summary, this section has addressed Obj2 in analysing the relearning behaviour of Soilse. The *ExpFactor* plays an important role in determining the behaviour, i.e., relearning period and the decay rate, by which local Soilse agents respond upon detecting genuine traffic pattern changes. This directly affects the overall performance of Soilse in terms of AWT and AvgStops where best performance of the first does not guarantee a corresponding best performance of the latter. In addition, it is observed that

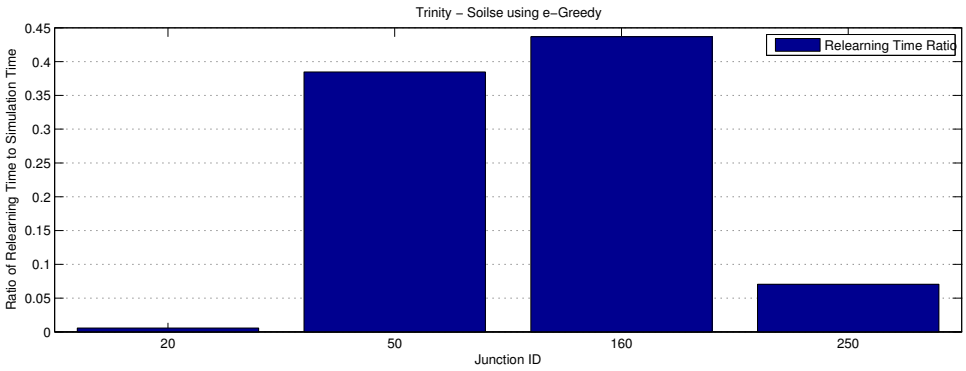


Figure 5.18: Trinity - Soilse using ϵ-greedy ratio of relearning time to simulation time - best AWT performance

certain Soilse agents are affected by local occurrences of genuine traffic pattern changes at different stages. Hence, each affected Soilse agent relearns differently.

5.3.3 SoilseC

Unlike a Soilse agent, a SoilseC agent allows for collaboration through information exchange among neighbouring agents in a given collaboration mode (see Section (3.5.2.1)). Here, we analyse the effect of collaboration by comparing the best performance of SoilseC against Soilse's best performance. We also study the effect of multiple collaboration modes on the overall performance in terms of AWT and AvgStops. We also analyse the effect of the frequency at which collaboration occurs in a given mode. Overall, this section is presented to address Obj4.

5.3.3.1 SoilseC vs. Soilse

The effect of collaboration in SoilseC on the overall performance in terms of AWT and AvgStops is studied against non-collaborative Soilse (see Obj4). Collaboration in SoilseC implies that each SoilseC agent is influenced by its neighbours' performance depending on the collaboration mode in effect.

Table (5.7) summarizes the best performance of SoilseC versus Soilse in terms of AWT. SoilseC using different action selection strategies varied in their performance compared to their Soilse counterparts. SoilseC using Boltzmann showed the best improvement in terms of AWT by reducing the latter by $\sim -4.78\%$ as opposed to its Soilse counterpart. This is due to the nature of Boltzmann that bases its action selection on a probability model built using the policy's different Q-values. The latter in the case of SoilseC are more well-informed about the consequences of their associated actions through the exchanged history of discounted and normalized rewards of certain neighbours. However,

	Soilse			SoilseC			SoilseC Performance∓		
	∼AWT⋆	#AV	∼AvgStops	∼AWT⋆	#AV	∼AvgStops	∼AWT%	#AV%	∼AvgStops%
ϵ-greedy	30.484	30152	4.37	**30.009**	**30347**	**3.91**	-1.58%	+0.64%	-10.5%
Greedy	33.345	29651	4.47	32.287	29570	4.75	-3.17%	-0.27%	+5.89%
Boltzmann	44.864	29448	5.74	42.719	29466	5.47	-4.78%	+0.06%	-4.70%

Table 5.7: Trinity - SoilseC vs. Soilse for best performance based on AWT

	Soilse			SoilseC			SoilseC Performance∓		
	∼AWT⋆	#AV	∼AvgStops	∼AWT⋆	#AV	∼AvgStops	∼AWT%	#AV%	∼AvgStops%
ϵ-greedy	31.765	29597	4.17	**30.009**	**30347**	**3.91**	-5.52%	+2.47%	-6.23%
Greedy	33.345	29651	4.47	33.712	30128	4.41	+1.08%	+1.58%	-1.34%
Boltzmann	45.369	29285	5.59	43.725	29384	5.44	-3.62%	+0.33%	-2.68%

⋆Results are in seconds.

∓The negative sign (-) indicates lower AWT, AvgStops or #Arrived Vehicles (AV) , otherwise (+) is used.

Table 5.8: Trinity - SoilseC vs. Soilse for best performance based on AvgStops

analogous to the Soilse situation, SoilseC using Boltzmann regardless of its enhanced performance through collaboration, did not provide competitive AWT performance against SoilseC using other action selection strategies. Furthermore, SoilseC using ϵ-greedy and greedy provide $\sim -1.58\%$ and $\sim -3.17\%$ better performance in terms of AWT as opposed to their Soilse counterparts. These AWT performance results are not notable, however, the effect of collaboration was clear in terms of the AvgStops metric in the case of the best performing SoilseC using ϵ-greedy, specifically, $\sim -10.52\%$ better performance in terms of AvgStops as opposed to Soilse using ϵ-greedy. The overall difference in terms of the number of arrived vehicles was minimal in all SoilseC cases compared to Soilse.

Table (5.8) summarizes the best performance of SoilseC vs. Soilse in terms of AvgStops. It is noticeable that the best AvgStops performance for SoilseC using ϵ-greedy is also the best performance in terms of AWT. Collaboration has resulted in a balance between SoilseC's performance in terms of both metrics as opposed to Soilse using ϵ-greedy case. As far as the best performance of SoilseC using ϵ-greedy is concerned, a $\sim -5.52\%$ and $\sim -6.23\%$ better performance in terms of AWT and AvgStops results respectively compared to Soilse. Moreover, an adverse effect of collaboration for the best AvgStops performance for SoilseC using greedy in terms of AWT is noticed, i.e., $\sim +1.08\%$ higher AWT than in the case of Soilse. Similar adverse effect of collaboration can be noticed in SoilseC using greedy in terms of AvgStops as it resulted in $\sim +5.89\%$ more AvgStops that the counterpart case of Soilse. It appears that in both cases of best performing SoilseC using greedy, each case provides better performance than the respective Soilse solely on one main metric (AvgStops or AWT) at a time. This implies that SoilseC using greedy cannot balance between achieving best performance in terms

	Best AvgStops Performance		Best AWT Performance	
	ExpFactor	γ	ExpFactor	γ
e-greedy	2	0.7	5	0.7
Greedy	2	0.3	2	0.3
Boltzmann	2	0.7	5	0.3

Table 5.9: Trinity - key parameters of best performing Soilse

	Best AvgStops Performance				Best AWT Performance			
	ExpFactor	CollFreq (seconds)	CM*	γ	ExpFactor	CollFreq (seconds)	CM*	γ
e-greedy	2	240	Three	0.3	2	240	Three	0.3
Greedy	2	240	Two	0.7	2	240	Tow	0.3
Boltzmann	2	120	Three	0.7	2	240	Tow	0.3

*Collaboration Mode

Table 5.10: Trinity - key parameters of best performing SoilseC

of AWT and AvgStops simultaneously.

Table (5.9) provides key parameters for the experimental setup of Soilse's best performance. It appears that the AvgStops performance is directly affected by the amount of exploration in Soilse as it performs best in terms of that metric under $ExpFactor\,2$ for all action selection strategies. We believe this is due to the sensitivity of that metric (due to breaking propagation to following vehicles) where it is more adversely affected (as opposed to AWT) by exploratory actions taken over longer exploration durations. On the other hand, Soilse using ϵ-greedy, which has the best AWT performance, required more exploration, i.e., $ExpFactor\,5$ in order to perform best. Soilse using Boltzmann had also a similar situation. However, Soilse using greedy, having a non-exploratory action selection (only Q-learning's α controls the exploration) performed best in terms of AWT and AvgStops under $ExpFactor\,2$ and $\gamma = 0.3$. In terms of γ, Soilse using ϵ-greedy performed best by being more farsighted under $\gamma = 0.7$ while in the case of greedy, as expected, nearsightedness using $\gamma = 0.3$ performed best.

Concerning SoilseC, Table (5.10) provides key parameters for SoilseC's best performance experimental setup. It is noticeable that best SoilseC performance in terms of AWT and AvgStops using different action selection strategies required lower exploration periods given $ExpFactor\,2$. This highlights the advantage of collaboration in SoilseC in terms of less required relearning durations in its outperformance of Soilse (see Figures (5.18)(5.23)). Furthermore, given that the best performance in terms of AWT and AvgStops was for the same SoilseC using ϵ-greedy, naturally the same parameters are shared. The majority of SoilseC using different action selection strategies performed best under a collaboration frequency of $240s$ as opposed to more frequent $120s$. This is due to more informative (longer) exchanged history using $CollFreq = 240s$.

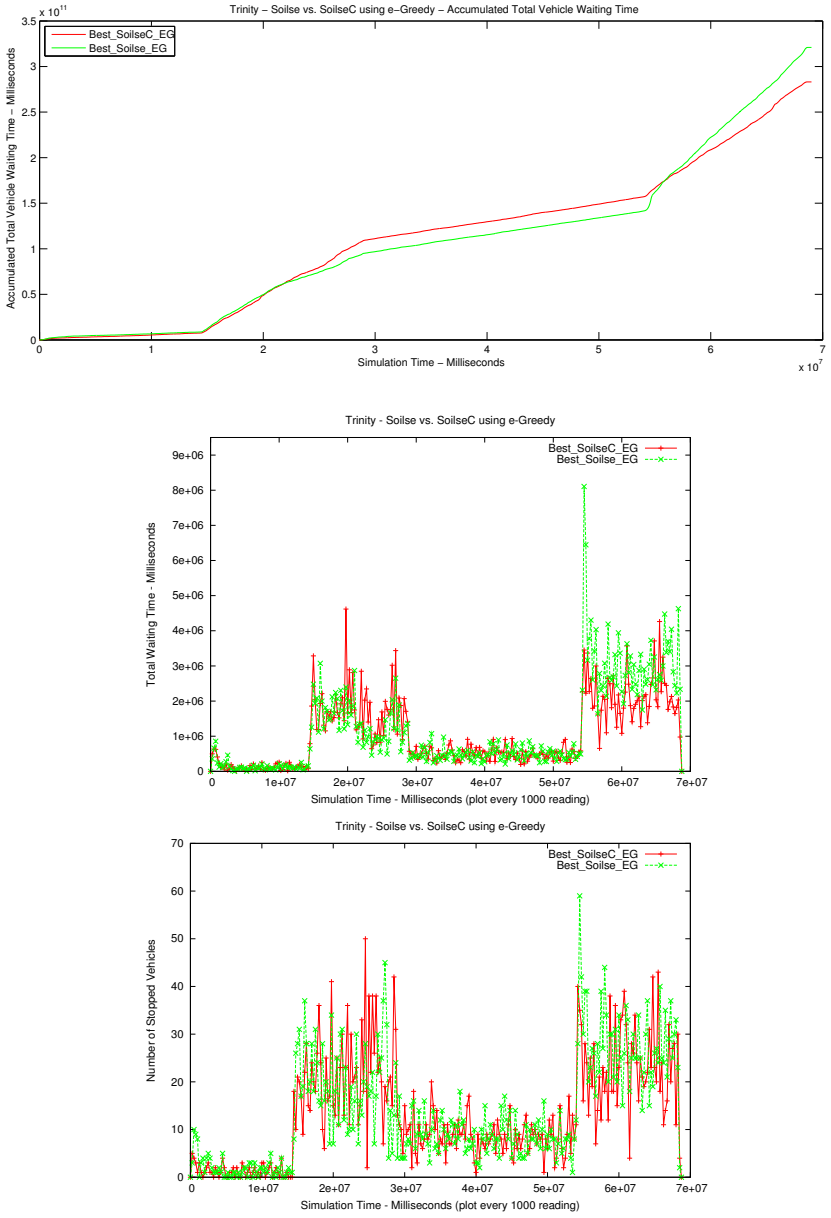


Figure 5.19: Trinity - Soilse vs. SoilseC using ϵ-greedy - accumulated total vehicle waiting time, total vehicle waiting and number of stopped vehicles throughout the simulation time - best AWT performance

Graphs presented in Figure (5.19) reaffirm SoilseC using ϵ-greedy's generally better performance in terms of AWT and AvgStops against Soilse as discussed earlier. It appears that SoilseC in that case had certain expensive relearning tasks during the MPP that resulted in a relatively lower performance in terms of total vehicle waiting time during the second half of the MPP. However, SoilseC considerably outperformed Soilse during the EPP in terms of AWT and AvgStops as it exhibited an efficient relearning manner that reached a better policy than Soilse's. The accumulation of total waiting time in SoilseC against Soilse confirms that observation as Soilse's accumulated total waiting time surges beyond SoilseC's at the beginning of the EPP.

As far as SoilseC using Boltzmann is concerned, the graphs presented in Figure (5.20) confirm its overall better performance against Soilse using Boltzmann in terms of AWT and AvgStops. SoilseC maintained a lower total waiting time and lower relearning cost (characterized by the absence of severe surges in total waiting time) than Soilse in the EPP. This is reaffirmed through the lower accumulation of total vehicle waiting time. Concerning the number of stopped vehicles throughout the simulation, SoilseC maintained a lower number of stopped vehicles especially during the MPP and the EPP.

Observing the graphs of SoilseC using greedy against the corresponding Soilse do not reveal major differences in performance as presented in Figure (5.21). This reaffirms the comparison between their best AWT and AvgStops results that was discussed earlier. However, it appears that SoilseC performed better in terms of the number of stopped vehicles especially during the EPP.

In summary, the performance of SoilseC versus Soilse performance was assessed (see Obj4). SoilseC using ϵ-greedy (under best AWT performance) clearly outperformed the Soilse counterpart in terms of AvgStops while this was the case in both terms of AWT and AvgStops under the best AvgStops performance. On the other hand, SoilseC using Boltzmann outperformed the Soilse counterpart in all cases while generally there was no improvement in SoilseC using greedy performance against the Soilse counterpart. Essentially, the effect of collaboration on SoilseC's performance against Soilse was moderate given this scenario's scale.

5.3.3.2 Collaboration Mode

Here we study the effect of the collaboration mode (CM) on SoilseC's performance (see Obj4). We choose SoilseC using ϵ-greedy to compare among its best AWT performance per collaboration mode, see Table (5.11).

The best performing SoilseC deployments under all collaboration modes used a collaboration frequency of 240 seconds while the *ExpFactor* value varied to 1 in CM one, 5 in CM two and 2 in CM three. It appears that CM three performed best in terms of AWT, AvgStops and the number of

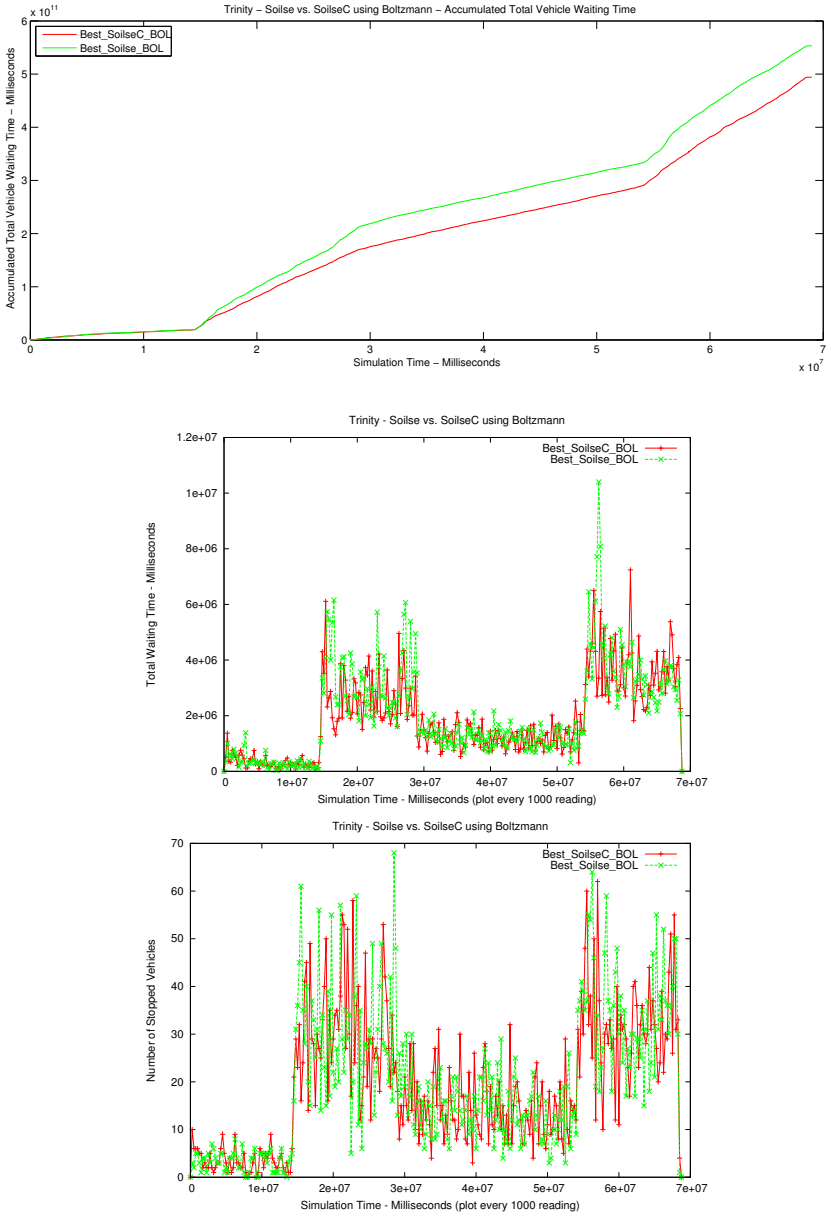


Figure 5.20: Trinity - Soilse vs. SoilseC using Boltzmann - accumulated total vehicle waiting time, total vehicle waiting and number of stopped vehicles throughout the simulation time - best AWT performance

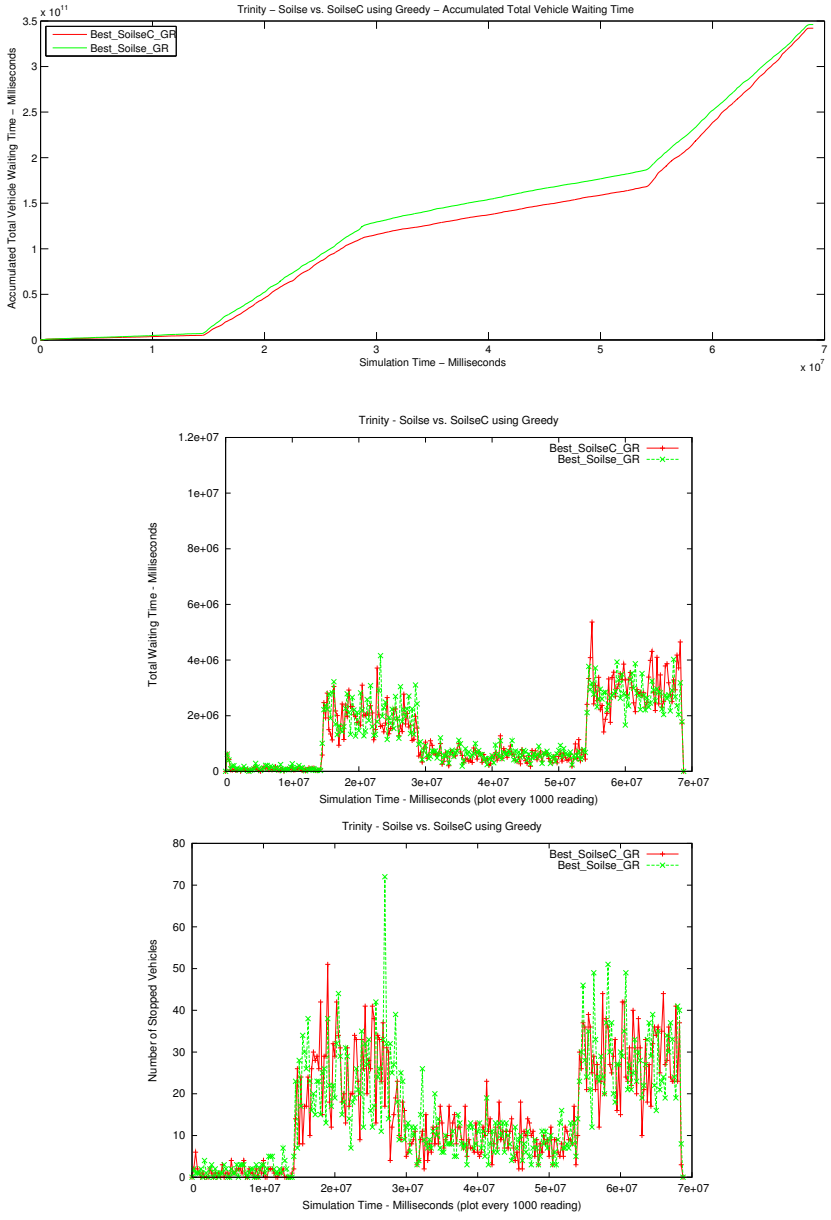


Figure 5.21: Trinity - Soilse vs. SoilseC using greedy - accumulated total vehicle waiting time, total vehicle waiting time and number of stopped vehicles throughout the simulation time - best AWT performance

ϵ-greedy / Collaboration Mode	∼AWT (seconds)	#Arrived Vehicles	∼AvgStops
One	30.338	29574	4.07
Two	31.038	30076	4.27
Three	30.009	30347	3.91

Performance %	Collaboration Mode Three		
	∼AWT%	∼#Arrived Vehicles%	∼AvgStops%
Collaboration Mode One	-1.08%	+2.54%	-3.93%
Collaboration Mode Two	-3.31%	+0.89%	-8.43%

Table 5.11: Trinity - SoilseC best AWT performance per collaboration mode - ϵ-greedy

vehicles arrived. Its approach of allowing all neighbours, regardless of being upstream of downstream, to share their recent performance with a given SoilseC agent appears beneficial in this scenario. It performed better against CM two in terms of AvgStops by providing $\sim -8.43\%$ less AvgStops. However, in terms of AWT the difference between CM three and CM two was a moderate $\sim -3.31\%$ lower AWT from the CM three side. The latter performed also moderately better against CM one in terms of AWT and AvgStops by providing $\sim -1.08\%$ and $\sim -3.93\%$ better performance respectively. In terms of the number of arrived vehicles, CM three provided $\sim +2.54\%$ more vehicles against CM one while the difference was minimal against CM two.

In summary, the effect of the collaboration mode on SoilseC was assessed given this scenario's scale as required in Obj4. In addition, it is difficult at this stage given the current scenario's scale to see a noticeable difference in the overall performance of different CMs. However, CM one and CM three appeared to have a close performance.

5.3.3.3 Relearning Behaviour

This section addresses Obj2 concerning relearning in SoilseC. The best performing SoilseC using ϵ-greedy is considered. The ratio of the relearning time to the simulation time (see Figure (5.23)) and relearning occurrences (see Figure (5.22)) are presented.

It can be observed that only two SoilseC agents controlling signalized junctions ID 50 and ID 160 have relearnt. Bearing in mind that this SoilseC best performance is at $ExpFactor = 2$, the relearning periods are relatively shorter than the ones seen in the Soilse case where $ExpFactor = 5$. It is clear that SoilseC at junction ID 50 was continuously affected by genuine traffic pattern changes during the MPP and towards the beginning of the UHP. On the other hand, SoilseC at junction ID 160 was affected during the MPP and during the EPP. Moreover, SoilseC agents at junctions ID 50 and ID 160 have spent $\sim 17\%$ and $\sim 25\%$ of the simulation time relearning respectively. In summary,

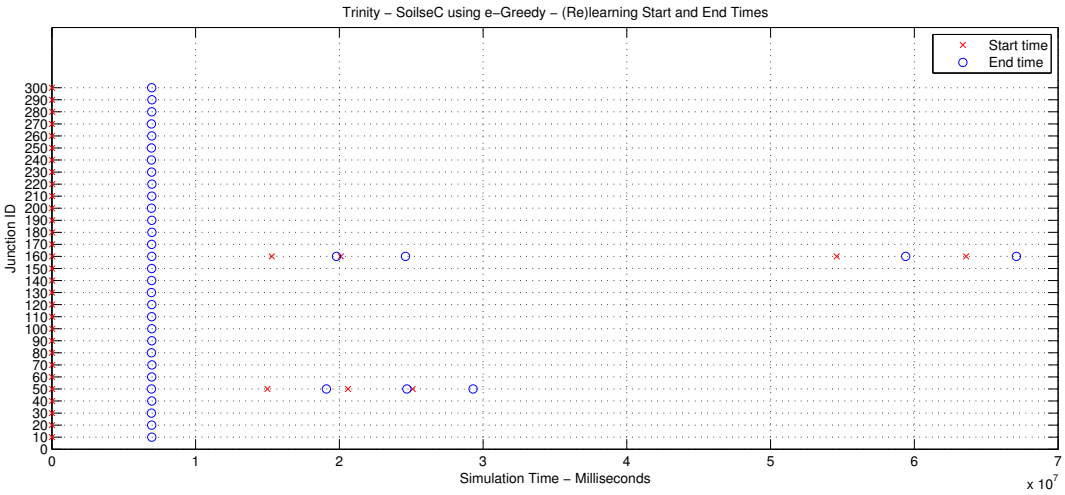


Figure 5.22: Trinity - SoilseC using ϵ-greedy (re)learning start and end times - best AWT performance

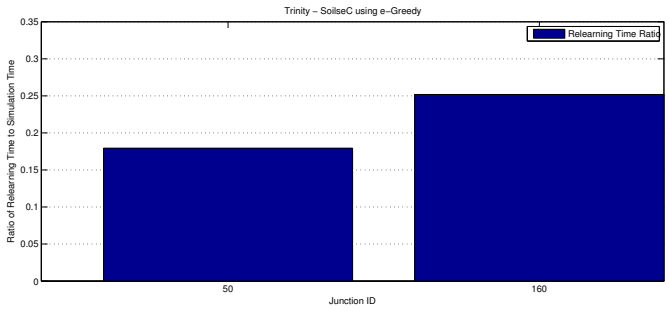


Figure 5.23: Trinity - SoilseC using ϵ-greedy ratio of relearning time to simulation time - best AWT performance

	Soilse*			SoilseC*			RR20s	SAT_2_1.5
	ϵ-greedy	Greedy	Boltzmann	ϵ-greedy	Greedy	Boltzmann		
∼AWT (seconds)	**30.484**	33.338	44.864	**30.009**	32.287	42.719	34.031	37.885
#Arrived Vehicles	**30152**	29561	29448	**30347**	29570	29466	29128	30162
∼AvgStops	**4.37**	4.42	5.74	**3.91**	4.75	5.47	4.78	4.9

Performance% Against (RR20s, SAT_2_1.5)	Soilse*		
	∼AWT%	∼#Arrived Vehicles%	∼AvgStops%
ϵ-greedy	(-10.42%, -19.53%)	(+3.39%, -0.03%)	(-8.57%, -10.81%)
Greedy	(-2.03%, -12.00%)	(+1.76%, -1.99%)	(-7.53%, -9.79%)
Boltzmann	(+24.14%, +15.55%)	(+1.08%, -2.36%)	(+16.72%, +14.63%)

Performance% Against (RR20s, SAT_2_1.5)	SoilseC*		
	∼AWT%	∼#Arrived Vehicles%	∼AvgStops%
ϵ-greedy	(-11.81%, -20.78%)	(+4.01%, +0.60%)	(-18.20%, -20.20%)
Greedy	(-5.12%, -14.77%)	(+1.49%, -1.96%)	(-0.627%, -3.061%)
Boltzmann	(+20.33%, +11.31%)	(+1.14%, -2.30%)	(+12.61%, +10.42%)

*Soilse and SoilseC results are calculated after the initial learning has elapsed.
(Best Soilse and SoilseC results are in bold font.)

Table 5.12: Trinity - best AWT performance of Soilse and SoilseC against the selected baselines

these are considerably lower ratios in comparison to the Soilse case and yet SoilseC outperformed Soilse in terms of AWT and AvgStops. In addition, two SoilseC agents experienced relearning periods as opposed to four in the Soilse case.

5.3.4 Comparison Against Baselines

This section presents a comparison between the best performing Soilse and SoilseC using each action selection strategies for the Trinity scenario against the previously selected baselines, i.e., RR20s and SAT_2_1.5. This aims at addressing Obj3. The performance comparison is presented in Table (5.12).

Results show that best performing Soilse and SoilseC using ϵ-greedy clearly outperform both baselines. While Soilse provides $\sim -10.42\%$ and $\sim -19.53\%$ better performance in terms of AWT against RR20s and SAT_2_1.5 respectively, SoilseC exceeds this performance by providing $\sim -11.81\%$ and $\sim -20.78\%$ lower AWT as opposed to RR20s and SAT_2_1.5 respectively. In terms of AvgStops performance, Soilse outperforms RR20s and SAT_2_1.5 by $\sim -8.57\%$ and $\sim -10.81\%$ less AvgStops respectively. On the other hand, SoilseC performance in terms of AvgStops is approximately twice more notable than Soilse's performance against the baselines, i.e., $\sim -18.20\%$ and $\sim -20.20\%$ less AvgStops against RR20s and SAT_2_1.5 respectively. Furthermore, Soilse and SoilseC performance in terms of the number of arrived vehicles was not clear against SAT_2_1.5 (bear in mind that the

metric is based on the number of vehicles inserted after the initial learning has finished). However, against RR20s, Soilse allowed $\sim +3.39\%$ more vehicles to arrive while SoilseC allowed $\sim +4.01\%$ more vehicles to do so.

Concerning Soilse and SoilseC using greedy, their overall performance in terms of AWT and AvgStops was not as good when compared to the cases where both used ϵ-greedy. Soilse using greedy reduced the AWT and AvgStops by $\sim -2.03\%$ and $\sim -7.53\%$ respectively against RR20s. However, its performance was better against SAT_2_1.5 as it reduced the AWT and AvgStops by $\sim -12.00\%$ and $\sim -9.79\%$ respectively. The difference in the number of arrived vehicles between Soilse using greedy and both baselines was marginal but better against RR20s. As far as SoilseC using greedy is concerned, it outperformed its Soilse counterpart in terms of AWT against RR20s and SAT_2_1.5, i.e., $\sim -5.12\%$ and $\sim -14.77\%$ respectively. However, SoilseC's performance using greedy in terms of AvgStops was poorer than its Soilse counterpart against both baselines. Similar to Soilse's performance, the difference in the number of arrived vehicles between SoilseC using greedy and both baselines was marginal but better against RR20s.

Soilse and SoilseC using Boltzmann as an action selection strategy in this scenario did not outperform the baselines. However, SoilseC using Boltzmann provided better performance than its Soilse counterpart in general but still remained insufficient against the baselines. Both Soilse and SoilseC using Boltzmann only managed to provide a marginally better performance in terms of the number of arrived vehicles against RR20s. It appears that Boltzmann did not manage to relearn given the period of time determined by some pattern change a suitable policy for the emerging pattern. This is believed to be due to the demanding nature of Boltzmann when it comes to the amount or exploration required to achieve a near optimal policy and to its seemingly expensive cost of relearning. In addition, this stems from the nature of Boltzmann that selects its actions based on a probability model built using the underlying policy. As a result, upon a pattern change detection, the action selection process is adversely affected by the existing policy until it is gradually overridden. This situation does not occur while using ϵ-greedy or greedy, given that the latter uses a fixed non-exploratory greedy action selection while ϵ-greedy follows a random action selection upon relearning relative to its newly set ϵ value. Also, ϵ-greedy's relearning cost is not as affected by the existing policy as Boltzmann given that it only selects a greedy action based on the probability of $1 - \epsilon$ which is independent from the existing policy.

Soilse and SoilseC using ϵ-greedy are selected, given their best overall performance against the baselines, for further comparison in terms of the ongoing (accumulated) total vehicle waiting time and the (accumulated) number of stopped vehicles against the baselines. Figure (5.24) clearly shows

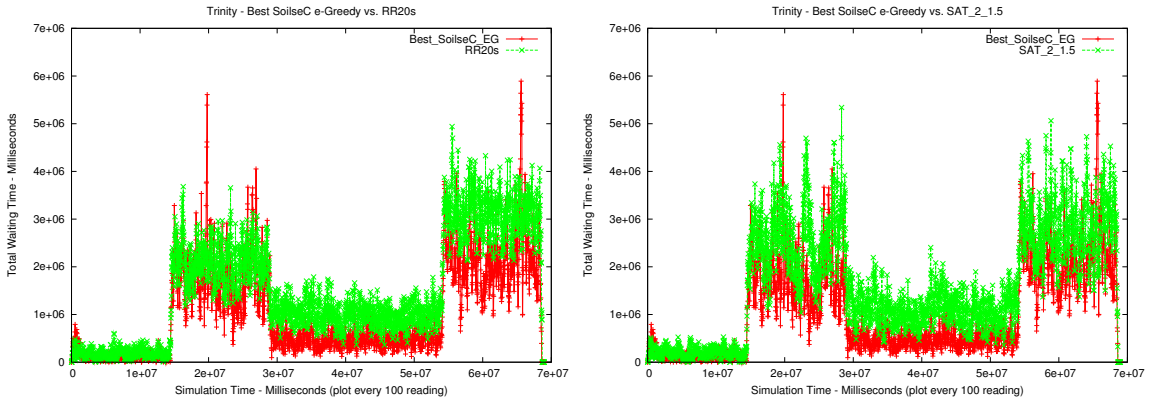


Figure 5.24: Trinity - SoilseC ϵ-greedy vs. (RR20s and SAT_2_1.5) - total vehicle waiting time throughout the simulation time - best AWT performance

the difference between the performance of SoilseC against both baselines in terms of total vehicle waiting time throughout the simulation time. SoilseC maintains a considerably lower total vehicle waiting time nearly all the time with brief exceptions during some relearning periods. Similarly, Soilse using ϵ-greedy, see Figure (5.25), shows a clear lower total vehicle waiting time through the simulation against both baselines with the exception of the performance during the EPP in the case of comparison against SAT_2_1.5 where it maintains slightly similar performance. To a certain degree this is true against RR20s but with better performance from the Soilse side at several points in time during the EPP. For more clarification on the ongoing performance of Soilse, SoilseC and the baselines, Figure (5.26) shows a comparison for the accumulation of the total vehicle waiting time. It is clear that SAT_2_1.5 performs poorly in this scenario against the other selected baseline and both Soilse and SoilseC. The difference in performance starts to get clearer in terms of the accumulated total vehicle time against RR20s mainly towards the end of the MPP. SoilseC maintains a lower trend in that case until the end as it relearned adequately while Soilse slightly deteriorates in performance towards the end as its relearning did not result in competitive policy taking into account the cost incurred.

In terms of the number of stopped vehicles throughout the simulation time, Figure (5.27) shows a comparison in that regard for SoilseC using ϵ-greedy against both baselines in addition to the graphs of the accumulation of the number of stopped vehicles. It can be observed that peaks in the number of stopped vehicles are more frequent in RR20s and SAT_2_1.5 cases as opposed SoilseC. This can be seen more clearly in the comparison of the accumulated number of stopped vehicles in the accompanying graphs. SoilseC maintained a lower trend against SAT_2_1.5 almost all the time while in the case of RR20s it appeared more notably after the MPP.

In the case of Soilse using ϵ-greedy, Figure (5.28) shows a comparison in terms of the number

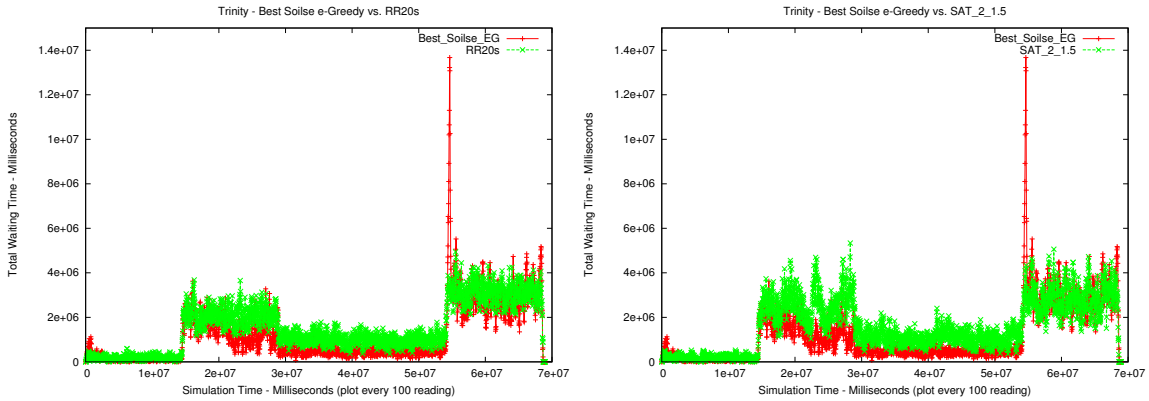


Figure 5.25: Trinity - Soilse ϵ-greedy vs. (RR20s and SAT_2_1.5) - total vehicle waiting time throughout the simulation time - best AWT performance

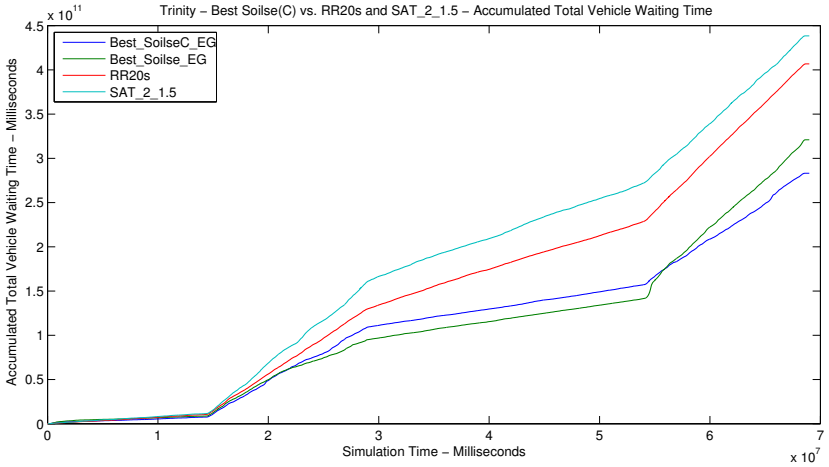


Figure 5.26: Trinity - Soilse and SoilseC ϵ-greedy vs. (RR20s and SAT_2_1.5) - accumulated total vehicle waiting time throughout the simulation time - best AWT performance

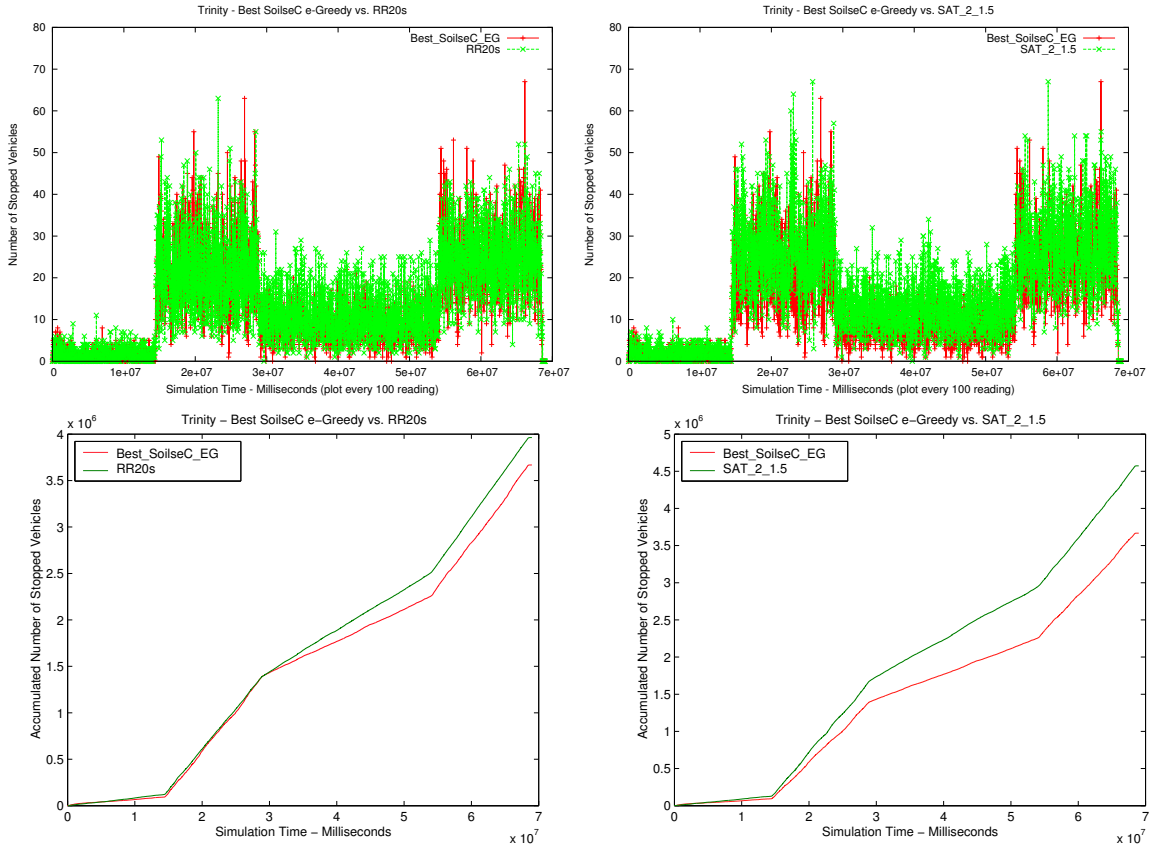


Figure 5.27: Trinity - SoilseC ϵ-greedy vs. (RR20s and SAT_2_1.5) - (accumulated) number of stopped vehicles throughout the simulation time - best AWT performance

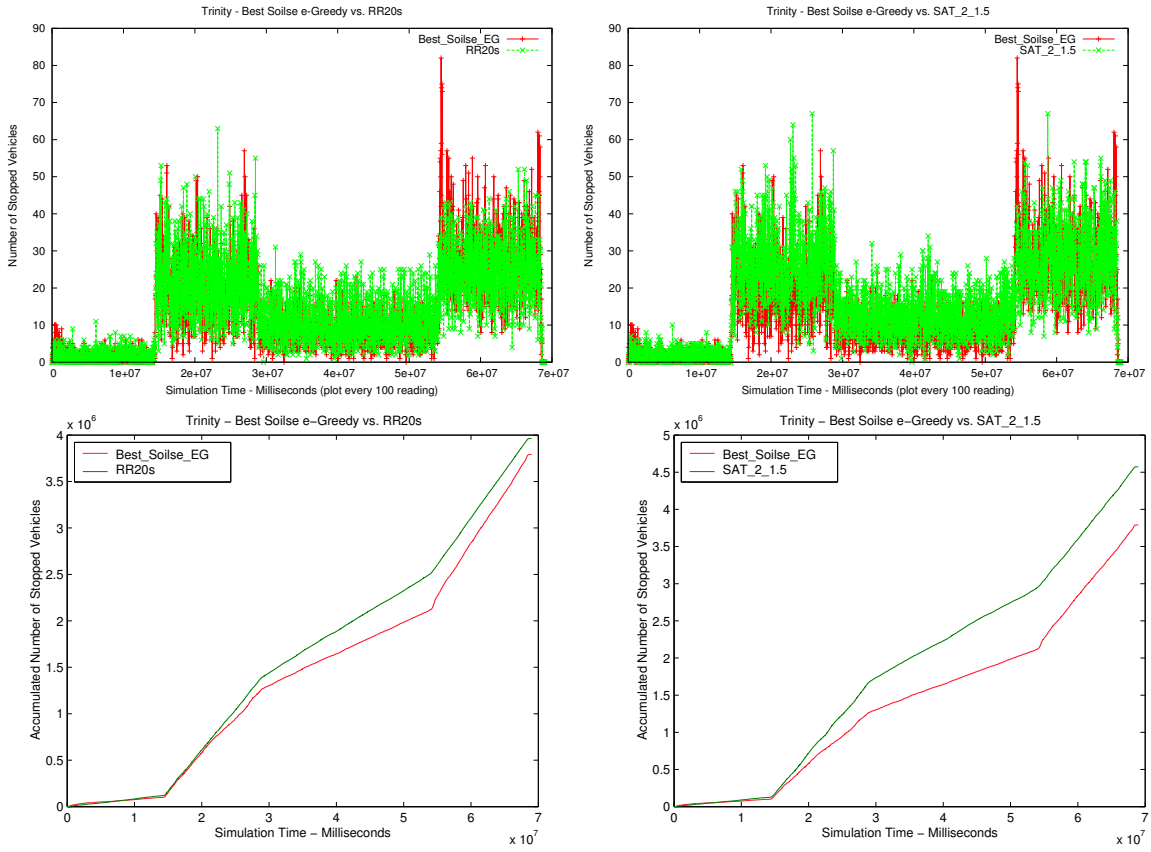


Figure 5.28: Trinity - Soilse ϵ-greedy vs. (RR20s and SAT_2_1.5) - (accumulated) number of stopped vehicles throughout the simulation time - best AWT performance

of stopped vehicles throughout the simulation time against both baselines in addition to the graphs of the accumulated number of stopped vehicles. It appears that Soilse maintains a consistently low trend against SAT_2_1.5 as far as the accumulation of the number of stopped vehicles is concerned nearly all the time. However, Soilse against RR20s maintains a lower trend given the accumulated number of stopped vehicles, but appears to deteriorate in performance in that regard during the EPP as it approaches RR20s performance towards the end. This is due to an expensive relearning period (notice surges towards the end in the number of stopped vehicles in Figure (5.28)) that Soilse initiates near the simulation end (without prospects of exploitation) as a response to the fading away of EPP. This situation is reminiscent of Soilse's performance in terms of the accumulated vehicle waiting time discussed earlier.

In summary, Soilse and SoilseC using ϵ-greedy as an action selection strategy have outperformed both RR20s and SAT_2_1.5 in terms of AWT and AvgStops hence satisfying Obj3. Notably, SoilseC

using ϵ-greedy provided approximately twice as good performance in terms of AvgStops as opposed to Soilse when compared to both baselines' performance (see Obj4). Soilse using greedy provided a better performance against SAT_2_1.5 in terms of AWT and AvgStops as opposed to its performance against RR20s. However, SoilseC using greedy outperformed the Soilse counterpart in terms of AWT against both baselines. Concerning the number of arrived vehicles, Soilse and SoilseC using ϵ-greedy provided a better performance against RR20s. However, the differences in other cases were marginal in that respect. In addition, SoilseC and Soilse using Boltzmann as an action selection strategy did not outperform both baselines in this scenario. However, SoilseC provided better performance than Soilse in that case in general but still remained insufficient against the baselines. The reason behind, as we explained earlier, is mainly due to the nature of the action selection process in Boltzmann that was adverse under the conditions of this scenario.

5.3.5 Summary

We presented and analysed the performance results from the Trinity scenario. The latter used a real world map of the surroundings of Trinity College Dublin and consisted of 30 signalized junctions. The simulation was based on $\sim$ 19 hours of traffic representing four different traffic patterns representing two uniform traffic situations of different loads and two peak situations for the morning the evening rush hours. Concerning the baselines for comparison, we selected the two best performing RR and SAT settings, namely, RR20s and SAT_2_1.5.

Firstly, Soilse's performance given its relearning behaviour was compared against a situation were only an initial learning was used (see Obj1). In addition, Soilse's performance was discussed where an evaluation for Soilse using different action selection strategies was presented (see Obj1). It was clear that Soilse using ϵ-greedy outperformed the cases where Boltzmann or greedy were used. We clarified the reasons for that which are mainly due to the efficient nature ϵ-greedy exhibits in relearning better policies. We also clarified the effect of the *ExpFactor* value and the overall relearning behaviour and presented details of such on a selected signalized junction (see Obj2). Furthermore, Soilse's performance was compared against SoilseC's where the latter had an overall better performance especially while using ϵ-greedy in terms of the AvgStops (see Obj4). Interestingly, while Soilse did not manage to provide a simultaneous best performance in terms of AWT and AvgStops, SoilseC was successful in doing so as it balanced its best performance for both metrics (see Obj4). The parameters for the best performance were also presented and discussed. The effect of the collaboration mode on SoilseC's best performing ϵ-greedy was presented where CM three performed best but closely to CM one (see Obj4).

Finally, a comparison between the best performing Soilse and SoilseC against the selected baselines was presented (see Obj3). It transpires that Soilse and SoilseC using ϵ-greedy as an action selection strategy have clearly outperformed both RR20s and SAT_2_1.5 by in terms of AWT and AvgStops. Soilse using ϵ-greedy provided $\sim -10.42\%$ and $\sim -19.53\%$ better performance in terms of AWT against RR20s and SAT_2_1.5 respectively. While SoilseC provided $\sim -11.81\%$ and $\sim -20.78\%$ better performance in terms of AWT against RR20s and SAT_2_1.5 respectively. In terms of AvgStops, Soilse and SoilseC outperformed RR20s by $\sim -8.57\%$ and $\sim -18.20\%$ respectively. When compared to SAT_2_1.5 in terms of AvgStops, Soilse and SoilseC provided $\sim -10.81\%$ and $\sim -20.20\%$ better performance respectively. On the other hand, Soilse using greedy as an action selection strategy performed well in certain cases especially against SAT_2_1.5 while the SoilseC counterpart outperformed that performance in terms of AWT against both baselines. Results from both Soilse and SoilseC using Boltzmann against the baselines were poor in this scenario. The reasons behind such a performance in this scenario were discussed earlier.

5.4 Dublin Inner City Centre Scenario

This section evaluates the performance of Soilse, SoilseC and the baselines in the Dublin Inner City Centre Scenario (DublinICC) scenario. This scenario is of a bigger scale than the Trinity scenario and we aim at assessing how Soilse and SoilseC can scale while maintaining better performance than the baselines (see Obj5). We first present the performance of the baselines in order to select the best performing baselines for further comparison against the best performing Soilse and SoilseC deployments. We discuss the effect of relearning on the performance of Soilse by comparing it against a situation where an initial learning was only used. The relearning behaviour in Soilse and SoilseC is discussed as well. We also discuss SoilseC's performance per collaboration mode. In addition, we concentrate on the performance of exploratory action selection strategies hence the performance of Soilse and SoilseC using greedy are not discussed in this scenario.

5.4.1 Baselines Performance

SAT and RR are used as the baselines in this scenario as well. Table (5.13) shows their performance under different settings in terms of AWT, AvgStops and the number of arrived vehicles.

RR using all settings (20s, 30s, 40s) provides clearly poor performance compared to all SAT performance. It is evident that RR does not cope with the scale and the traffic loads this scenario exhibits. RR20s appeared to perform the best against RR30s and RR40s. However, SAT_2_1.5

Seconds	RR ∼AWT*	RR #Arrived Vehicles	RR ∼AvgStops
20s	**622.736**	**47895**	**151.53**
30s	1396.943	41940	208.49
40s	1821.550	34110	226.04

	SAT ∼AWT*	SAT #Arrived Vehicles	SAT ∼AvgStops
2_1.5	**111.889**	56337	**28.24**
2_1.1	112.554	55374	40.35
5_1.5	114.253	**56601**	31.08
5_1.1	149.586	53976	64.03

*Results are in seconds
(Best performance results are in bold font)

Table 5.13: DublinICC - baselines performance - RR and SAT

	SoilseInit ∼AWT*	SoilseInit #AV	SoilseInit ∼AvgStops	Soilse ∼AWT*	Soilse #AV	Soilse ∼AvgStops	Soilse Performance∓ ∼AWT%	Soilse Performance∓ #AV%	Soilse Performance∓ ∼AvgStops%
ϵ-greedy	130.190	55043	38.95	71.918	56385	16.94	-44.75%	+2.38%	-56.50%
Boltzmann	153.202	54576	46.86	106.141	55519	28.25	-30.71%	+1.69%	-39.71%

* Numbers are in seconds

∓The negative sign (-) indicates lower AWT, AvgStops or #Arrived Vehicles (AV), otherwise (+) is used.

Table 5.14: DublinICC - SoilseInit vs. Soilse - best AWT performance

being the best performing SAT deployment in terms of AWT and AvgStops, outperformed RR20s by $\sim -82.03\%$ and $\sim -81.36\%$ in terms of AWT and AvgStops respectively. This is due to the fact that RR lacks any form of adaptiveness and responsiveness which has clearly resulted in clearly poor performance at this larger scale. On the other hand, SAT in its different settings proved to be a more competitive baseline especially SAT_2_1.5. Hence, we select RR20s and SAT_2_1.5 to be the baselines for further comparisons against Soilse and SoilseC.

5.4.2 Initial Learning vs. Relearning

Here we address Obj1 by comparing against Soilse that naturally relearns and a situation where an initial learning is only used (SoilseInit). Table (5.14) presents a comparison between Soilse and SoilseInit.

It is clearly observed that Soilse by relearning in both cases using ϵ-greedy and Boltzmann have notably outperformed the SoilseInit counterparts in terms of AWT and AvgStops. The difference in terms of the number of arrived vehicles was marginal however Soilse performed better in that sense as well. Soilse using ϵ-greedy has remarkably outperformed its SoilseInit counterpart by $\sim -44.75\%$ and $\sim -56.50\%$ lower AWT and AvgStops respectively. A similar case is with Soilse using Boltzmann where $\sim -30.71\%$ and $\sim -39.71\%$ lower AWT and AvgStops resulted respectively in comparison to its SoilseInit counterpart. Consequently, Obj1 is strongly satisfied in this scenario.

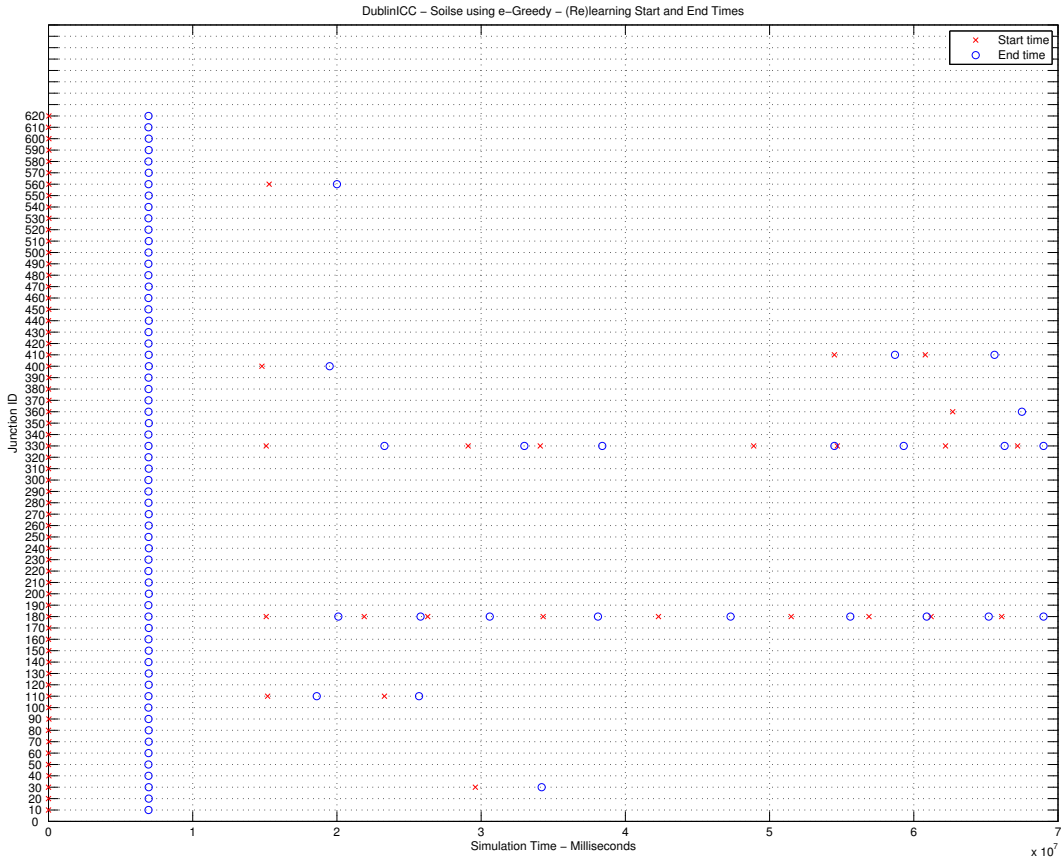


Figure 5.29: DublinICC - Soilse using ϵ-greedy (re)learning start and end times - best AWT performance

5.4.3 Relearning Behaviour

This section addresses Obj2 of analysing the relearning behaviour in the DublinICC scenario. We concentrate on discussing relearning occurrences and the ratio of relearning time to the simulation time. Both Soilse and SoilseC using ϵ-greedy are considered in their best AWT performance.

As far as Soilse is concerned, Figure (5.29) depicts the start and end times of learning and relearning periods for all Soilse agents controlling all signalized junctions. Two Soilse agents were more affected that the others in terms of relearning, namely, those controlling junctions ID 180 and ID 330. They needed to relearn on different points throughout the simulation period and during different traffic patterns. All of the affected Soilse agents appear to start relearning at the beginning of the MPP, however, only some (junctions IDs 180, 330 and 400) start relearning for the EPP and others (junctions IDs 180, 330 and 30) for the UHP. It can be noticed as well that some Soilse agents also relearn

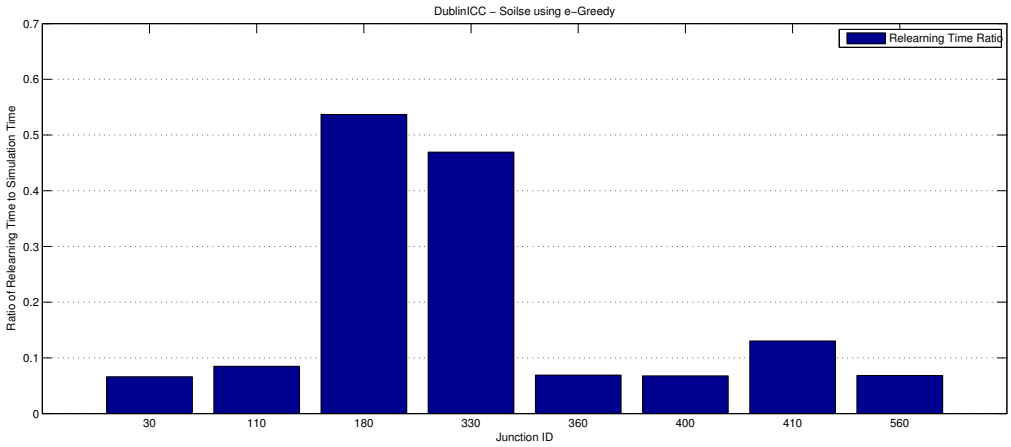


Figure 5.30: DublinICC - Soilse using ϵ-greedy ratio of relearning time to simulation time - best AWT performance

during some traffic patterns especially the Soilse agents at junctions IDs 180 and 330. This is due to continuous changes in their performance given their local traffic patterns. In Figure (5.30), the ratio of the relearning time to the simulation time for the eight affected Soilse agents is presented. It can be noticed that Soilse agents at junctions ID 330 and ID 180 have spent a considerable amount of time relearning relative to the simulation time, i.e., $\sim$ 53% and $\sim$ 46% respectively. However, it is worth mentioning again that relearning periods are not purely for exploration so the aforementioned relearning ratios indeed include an increasing exploitation by time.

As far as SoilseC is concerned, Figure (5.31) depicts the start and end times of learning and relearning periods for all Soilse agents controlling all signalized junctions. It is noticeable that more SoilseC agents were affected as opposed to the number of affected Soilse agents. However, these SoilseC agents appear to relearn less frequently, which may be due to collaboration that provided them with better policies resulting in better performance (see Section (5.4.4)). Figure (5.32) presents the ratio of relearning time to the simulation time per affected SoilseC agent. Thirteen SoilseC agents have experienced some relearning at different times. SoilseC agents at junction IDs 330, 180, 410 and 50 have relearnt the most showing $\sim$ 41%, $\sim$ 28%, $\sim$ 28% and $\sim$ 19% ratios of relearning time to simulation time respectively. These are overall considerably lower ratios when compared to the situation under Soilse.

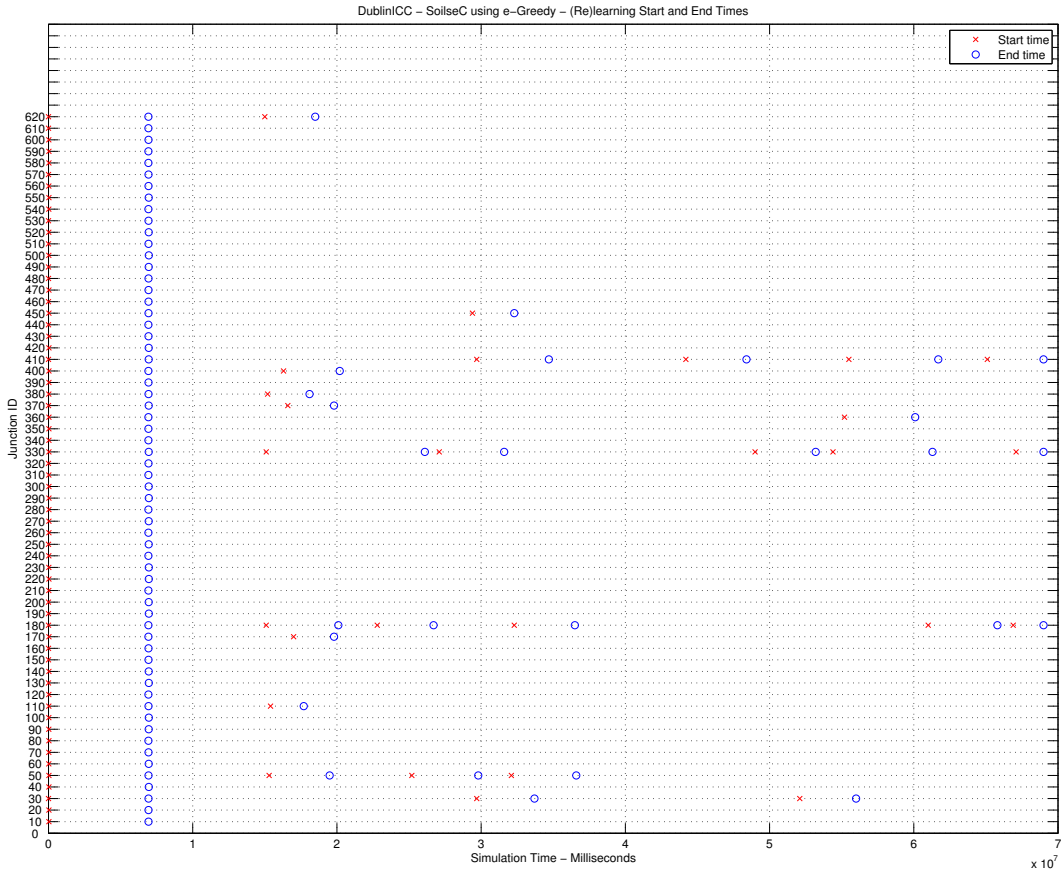


Figure 5.31: DublinICC - SoilseC using ϵ-greedy (re)learning start and end times - best AWT performance

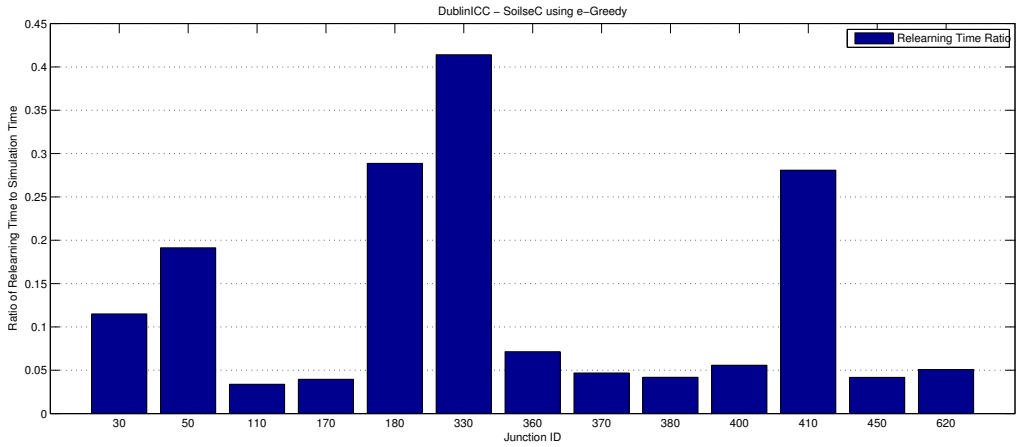


Figure 5.32: DublinICC - SoilseC using ϵ-greedy ratio of relearning time to simulation time - best AWT performance

	Soilse			SoilseC			SoilseC Performance$^{\mp}$		
	∼AWT*	#AV	∼AvgStops	∼AWT*	#AV	∼AvgStops	∼AWT%	∼#AV%	∼AvgStops%
ϵ-greedy	71.918	56385	16.94	63.471	56470	12.83	-11.74%	+0.15%	-24.26%
Boltzmann	106.141	55519	28.25	94.045	54853	24.91	-11.39%	-1.19%	-11.82%

*Results are in seconds.

$^{\mp}$The negative sign (-) indicates lower AWT, AvgStops or #Arrived Vehicles (AV) , otherwise (+) is used.

Table 5.15: DublinICC - SoilseC vs. Soilse best performance

5.4.4 Soilse and SoilseC vs. Baselines

We first discuss the performance of SoilseC against Soilse (see Obj4) by providing a comparison in terms of the AWT, AvgStops and the number of arrived vehicles. Table (5.15) presents the results of the best performing Soilse and SoilseC in this regard.

In this scenario, it was observed that both the Soilse and SoilseC that performed best in terms of AWT also performed the best in terms of AvgStops. Hence, the results presented in this scenario for both Soilse and SoilseC reflect best performance in both AWT and AvgStops terms. It appears that SoilseC maintains a better overall performance than Soilse in both cases of the action selection strategies. SoilseC using ϵ-greedy clearly outperforms the Soilse counterpart by providing $\sim 11.74\%$ and $\sim -24.26\%$ lower AWT and AvgStops respectively. As far as SoilseC using Boltzmann as an action selection strategy is concerned, it provided $\sim 11.39\%$ and $\sim -11.82\%$ lower AWT and AvgStops as opposed to its Soilse counterpart respectively. Furthermore, that the difference in performance in terms of the number of arrived vehicles between SoilseC and Soilse is marginal. SoilseC reached

Performance% Against (RR20s, SAT_2_1.5)	Soilse		
	∼AWT%	∼#Arrived Vehicles%	∼AvgStops%
ϵ-greedy	(-88.45%, -35.72%)	(+15.05%, +0.08%)	(-88.82%, -40.01%)
Boltzmann	(-82.95%, -5.13%)	(+13.73%, -1.45%)	(-81.53%, -0.035%)

Performance% Against (RR20s, SAT_2_1.5)	SoilseC		
	∼AWT%	∼#Arrived Vehicles%	∼AvgStops%
ϵ-greedy	(-89.80%, -43.27%)	(+15.18%, +0.23%)	(-91.53%, -54.56%)
Boltzmann	(-84.89%, -15.94%)	(+12.68%, -2.63%)	(-83.56%, -11.79%)

Table 5.16: DublinICC - Soilse and SoilseC best performance against baselines' best performance

better policies (hence the better performance) on the larger scale as opposed to Soilse compared to the performance of SoilseC against Soilse in the smaller scale Trinity scenario. This shows the benefits of collaboration as the scale grow larger where the positive effect of collaboration becomes more notable.

A performance comparison of Soilse and SoilseC against the selected best performing baselines (see Obj3) is presented in Table (5.16). Soilse and SoilseC has notably outperformed RR20s on this larger scale scenario in terms of all metrics. Soilse using ϵ-greedy provided a clearly $\sim -88.45\%$ lower AWT where SoilseC in that case provided $\sim -89.80\%$ against RR20s. Similarly, Soilse and SoilseC using Boltzmann provided $\sim 82.95\%$ and $\sim -84.89\%$ lower AWT against RR20s respectively. Soilse and SoilseC using ϵ-greedy also provided a notably better performance against RR20s in terms of AvgStops, i.e., $\sim -88.82\%$ and $\sim -91.53\%$ respectively. Similarly, Soilse and SoilseC using Boltzmann provided $\sim -81.53\%$ and $\sim -83.56\%$ lower AWT against RR20s respectively. In terms of the number of arrived vehicles, Soilse and SoilseC allowed more vehicles to arrive to their destinations in comparison to RR20s. Specifically, Soilse and SoilseC using ϵ-greedy allowed $\sim +15.05\%$ and $\sim +15.18\%$ more vehicles to arrive to their destinations respectively when compared to RR20s. When it comes to Soilse and SoilseC using Boltzmann in that regard, they allowed $\sim +13.73\%$ and $\sim +12.68\%$ more vehicles to arrive to their destinations respectively when compared to RR20s.

The best-performing SAT deployment, i.e., SAT_2_1.5 provided a more competitive baseline than RR20s. However, Soilse and SoilseC also outperformed SAT_2_1.5 clearly in terms of both AWT and AvgStops but differences in the number of arrived vehicles were generally marginal against SAT_2_1.5. Soilse using ϵ-greedy and Boltzmann provided $\sim -35.72\%$ and $\sim -5.13\%$ lower AWT as opposed to SAT_2_1.5 respectively. On the other hand, SoilseC using ϵ-greedy and Boltzmann provided $\sim -43.27\%$ and $\sim -15.94\%$ lower AWT as opposed to SAT_2_1.5 respectively. Furthermore, Soilse's performance in terms of AvgStops was more clear in the ϵ-greedy case, i.e., $\sim -40.01\%$

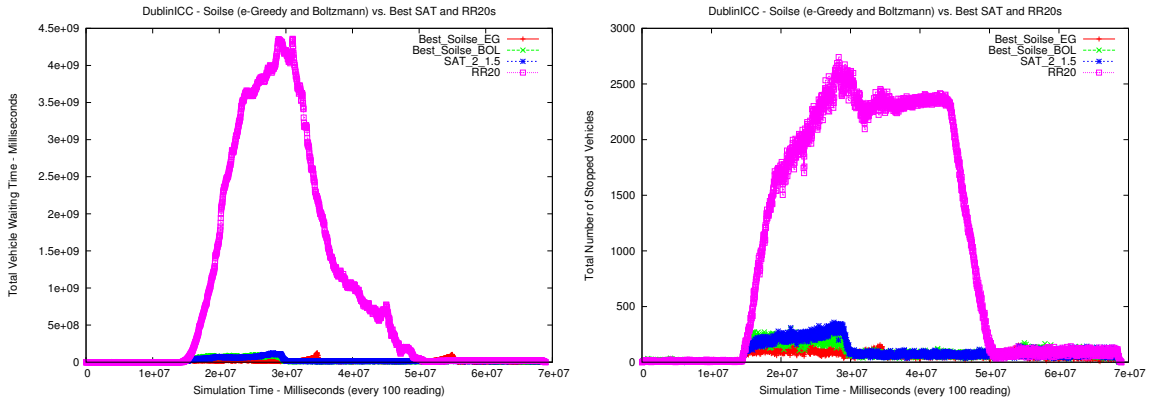


Figure 5.33: DublinICC - Best Soilse (ϵ-greedy and Boltzmann) performance vs. SAT and RR20s - total vehicle waiting time and total number of stopped vehicles throughout the simulation time

while in the case of Soilse using Boltzmann it was marginal against SAT_2_1.5. Concerning SoilseC's performance in terms of AvgStops, it provided a notable $\sim -54.56\%$ and $\sim -11.79\%$ less AvgStops as opposed to SAT_2_1.5 in cases of ϵ-greedy and Boltzmann respectively.

SAT_2_1.5 provided clearly better performance than RR20s overall and is a more competitive baseline to compare against in this scenario in terms of plots. Hence, we only show the performance of RR20s in the graphs presented in Figure (5.33) but exclude RR20s from further graphs given its out of scale performance. RR20s poor performance in terms of vehicle waiting time and number of stopped vehicles is clear in Figure (5.33). It caused a large total vehicle waiting time compared to best performing Soilse and SoilseC and SAT_2_1.5 that rendered theirs indistinguishable on the graph. Similarly the same adverse performance was noticeable in the number of stopped vehicles graph.

In order to compare the performance throughout the simulation time, Figure (5.34) presents the total vehicle waiting time and the total number of stopped vehicles graphs. These graphs compare against best Soilse and SoilseC performance in both ϵ-greedy and Boltzmann cases against SAT_2_1.5 and reaffirm the results discussed in Table (5.16). It can be observed that Soilse using ϵ-greedy showed a higher relearning cost (see surges in the first graph in Figure (5.34)) in terms of total vehicle waiting time and to a certain extent in terms of the total number of stopped vehicles as opposed to its SoilseC counterpart. Accordingly, collaboration in SoilseC resulted in a lower relearning cost and provided better performance than non-collaborative Soilse. Moreover, Soilse using Boltzmann did not provide a notably different performance than SAT_2_1.5. However, its SoilseC counterpart clearly outperforms SAT_2_1.5 during the MPP. Also, it appears that SoilseC using ϵ-greedy maintains a lower total vehicle waiting time and total number of stopped vehicles throughout the simulation against SAT_2_1.5 and SoilseC using Boltzmann. In addition, Soilse and SoilseC using Boltzmann

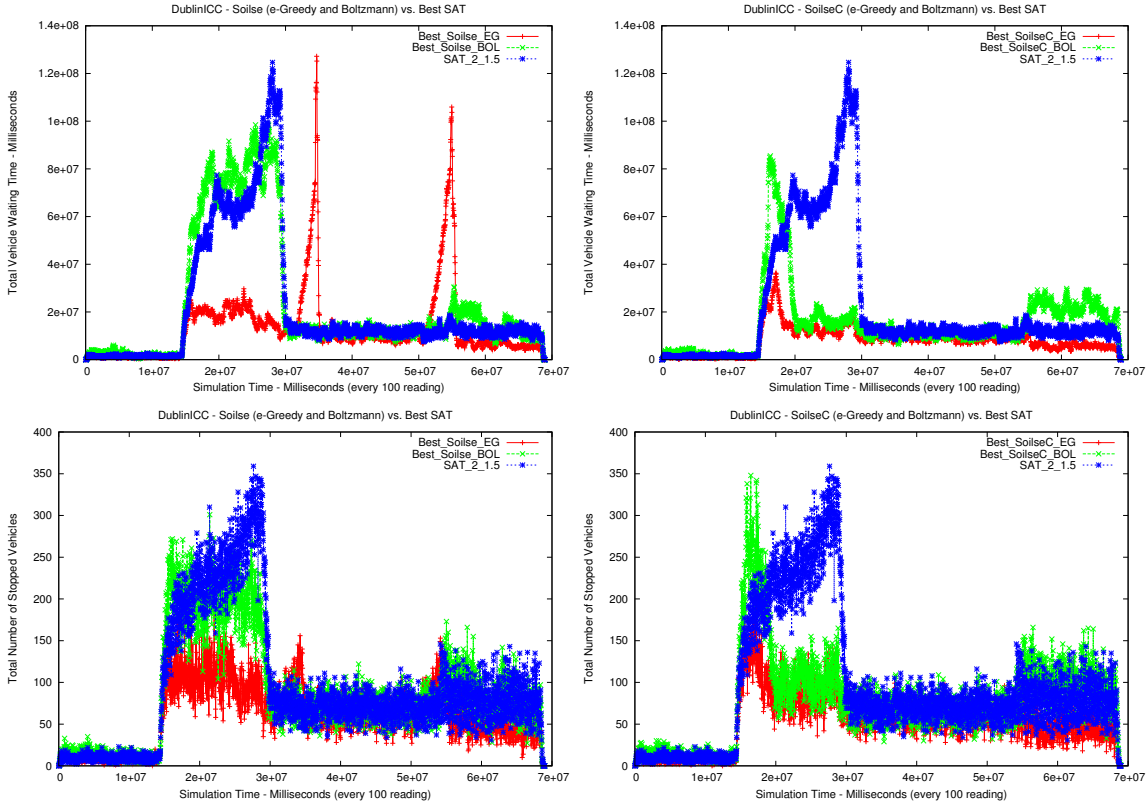


Figure 5.34: DublinICC - Soilse and SoilseC best performance vs. SAT - total vehicle waiting time and total number of stopped vehicles throughout the simulation time

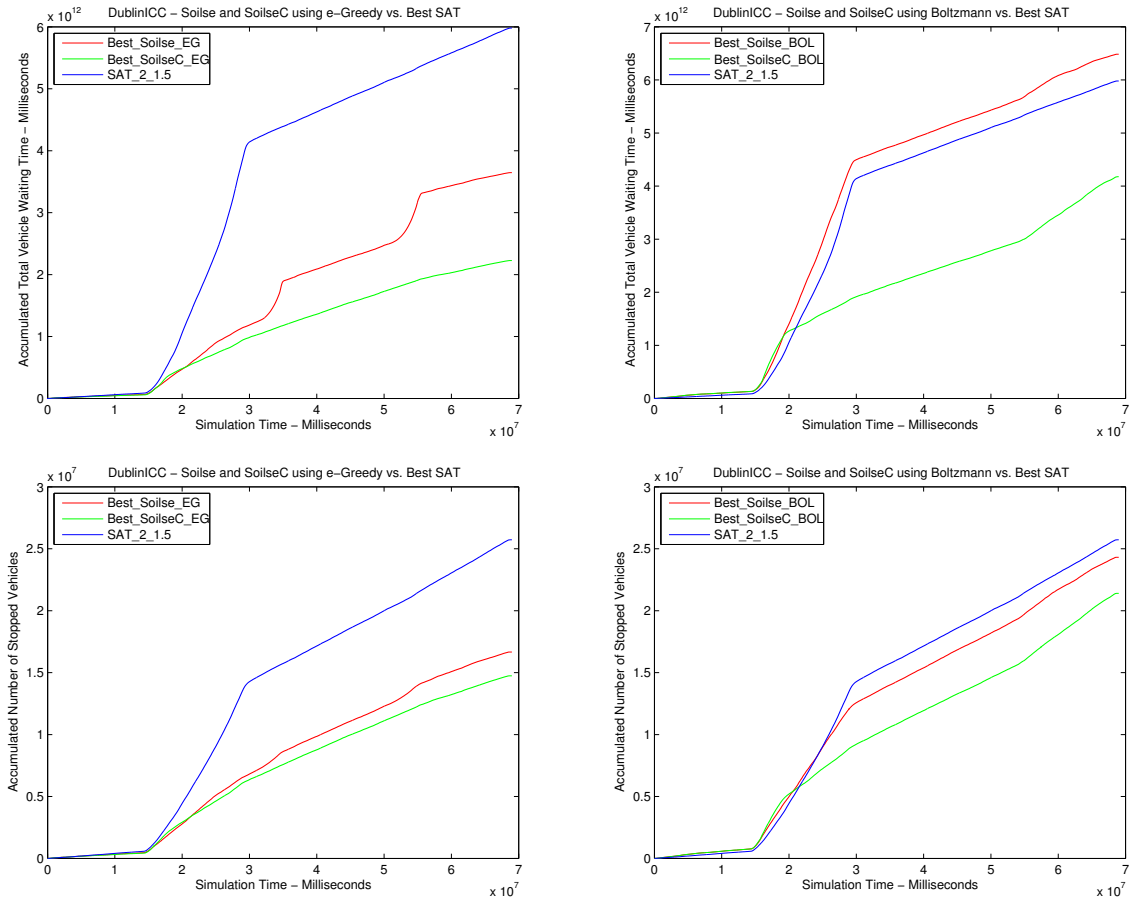


Figure 5.35: DublinICC - Soilse and SoilseC best performance vs. SAT - accumulated total vehicle waiting time and accumulated total number of stopped vehicles throughout the simulation time

do not reach a policy that allows them to perform notably better than SAT_2_1.5 during the EPP in terms of the total waiting time. This is in contrary to the cases where Soilse and SoilseC each use ϵ-greedy.

For a clearer view on the different performance throughout the simulation time, Figure (5.35) presents comparison graphs for the accumulated total vehicle waiting time and the accumulated total number of stopped vehicles regarding the best performing Soilse and SoilseC using ϵ-greedy against SAT_2_1.5. It is clear that SoilseC, regardless of the action selection strategy used, always maintains lower accumulated total vehicle waiting time and total number of stopped vehicles. Also, it can be noticed that the difference in both accumulations between SoilseC using ϵ-greedy against SAT_2_1.5 is bigger to the advantage of SoilseC than in the case where SoilseC uses Boltzmann. Concerning Soilse using ϵ-greedy, the relearning cost in terms of total vehicle waiting time that was exhibited in Figure

ϵ-greedy / Collaboration Mode	∼AWT (seconds)	#Arrived Vehicles	∼AvgStops
One	**63.471**	**56470**	**12.83**
Two	81.000	56419	22.60
Three	76.137	56450	18.81

Performance %	Collaboration Mode One		
	∼AWT%	∼#Arrived Vehicles%	∼AvgStops%
Collaboration Mode Two	-21.64%	+0.09%	-43.23%
Collaboration Mode Three	-16.63%	+0.03%	-31.79%

Table 5.17: DublinICC - SoilseC best performance per collaboration mode - ϵ-greedy

(5.34) is also clearly reflected on the accumulated total vehicle waiting time. On the other hand, SoilseC showed a more stable relearning, (i.e., without causing severe surges) in both total vehicle waiting time and total number of stopped vehicles accumulation. Moreover, as far as the performance of Soilse using Boltzmann in terms of AWT and AvgStops is concerned, it appears that it maintained a close accumulation trend to SAT_2_1.5 in terms of total vehicle waiting time and total number of stopped vehicles.

In summary, Soilse and SoilseC, both in cases of using ϵ-greedy and Boltzmann, have notably outperformed RR20s which performed poorly under this larger scale scenario. This result was expected given the inflexibility of RR to scale up and its lack of adaptiveness and responsiveness. On the other hand, best performing SAT, i.e., SAT_2_1.5 provided a more competitive baseline performance compared to RR20s. However, Soilse and SoilseC especially using ϵ-greedy as an action selection strategy have notably outperformed SAT_2_1.5 in all measured terms. Moreover, the overall performance of Soilse using Boltzmann against SAT_2_1.5 did not provide a notable difference. In addition, the differences between different Soilse and SoilseC performance in terms of the number of arrived vehicles against SAT_2_1.5 were marginal however notable against RR20s.

5.4.4.1 SoilseC's Collaboration Mode

The effect of the collaboration mode (CM) on SoilseC's performance using ϵ-greedy is assessed (see Obj4). Table (5.17) presents a performance comparison for SoilseC using different CMs. All best-performing SoilseC using ϵ-greedy and different CMs were using a collaboration frequency $CollFreq = 240s$. This appears to be a suitable frequency given the larger depth of performance history information exchanged as opposed to the case where $CollFreq = 120s$.

It appears that the best SoilseC performance is for SoilseC using CM one where a given SoilseC agent only sends its recent performance history to its downstream neighbours and receives such infor-

mation from the upstream ones. This is a logical way of collaboration given the traffic flow direction from the upstream junction towards the downstream junction that aligns with the logic of CM one. Such a logic serves as an early notifier to downstream junctions of a different possible patterns of incoming traffic through performance information sent from the upstream junction. The results from SoilseC using CM one are likely to be due to the extensive one-way streets system followed within Dublin inner city centre which aligns with the nature of CM one. Moreover, SoilseC using CM one outperformed SoilseC using CMs two and three by providing $\sim -21.64\%$ and $\sim -16.63\%$ lower AWT respectively. In terms of AvgStops, SoilseC using CM one outperformed SoilseC using CMs two and three by providing $\sim -43.23\%$ and $\sim -31.79\%$ lower AvgStops respectively.

Essentially, the underlying road network and the nature of traffic affect the performance resulting from the use of different CMs in SoilseC. In this scenario, it was clear that CM one performed best, while in the smaller scale scenario, i.e., Trinity scenario, it was not clearly evident.

5.4.5 Summary

This scenario presented results and comparisons concerning the performance of Soilse and SoilseC on the scale of Dublin inner city centre (see Obj5). A real map was used comprising 62 signalized junctions from an overall 270 junctions. The size of this map is comparable to the size of a city centre $\sim 62.6\%$ the size of Cork city centre.

Baselines performance analysis showed that RR20s and SAT_2_1.5 perform best among the different baselines settings. However, RR20s failed notably to scale up in this scenario as can be clearly seen in Figure (5.33). In addition, SAT_2_1.5 provided clearly better performance in all measured terms against RR20s and hence was used in comparison graphs against Soilse and SoilseC. Soilse by relearning has clearly outperformed SoilseInit hence satisfying Obj1. For example, Soilse using ϵ-greedy has remarkably outperformed its SoilseInit counterpart by $\sim -44.75\%$ and $\sim -56.50\%$ lower AWT and less AvgStops respectively. In addition, the relearning behaviour under Soilse and SoilseC was analysed (see Obj2).

Results from best performing Soilse and SoilseC from both action selection strategies have clearly outperformed both RR20s and SAT_2_1.5 in terms of AWT and AvgStops with the single exception of Soilse using Boltzmann in terms of AvgStops against SAT_2_1.5 (see Obj3). For example, SoilseC using ϵ-greedy outperformed RR20s in terms of AvgStops by $\sim -91.53\%$ and SAT_2_1.5 by $\sim -54.56\%$ hence satisfying Obj3. Moreover, Soilse and SoilseC using Boltzmann have generally provided a clearly better performance as opposed to the selected baselines in contrary to the situation under the Trinity scenario (taking into account the single exception mentioned earlier). In terms of the number

of arrived vehicles, performance differences between Soilse and SoilseC using both action selection strategies against SAT_2_1.5 were marginal however clear against RR20s.

Furthermore, a performance comparison among the different collaboration modes used in SoilseC was provided (see Obj4). It is noticed that the collaboration mode performance depends on the scale and nature of the scenario and its specifics. Collaboration mode one transpires to perform best under this scenario as it is believed to align with the extensive one-way streets nature of Dublin inner city centre.

Essentially, results from this higher scale scenario showed how both Soilse and Soilse scale and provide notably better performance against the baselines when compared to the smaller scale Trinity scenario and hence satisfying Obj5.

5.5 Summary

In this chapter we provided an evaluation based on two scenarios of different scales, Trinity College Dublin and the surroundings and Dublin inner city centre through which all the objectives mentioned in Section (5.2.5) were addressed individually. All signalized junctions in both scenarios used either Soilse or SoilseC agents at a time. For each scenario, two best performing baselines were selected to be compared against Soilse and SoilseC performance. These baselines were RR20s and SAT_2_1.5 in both scenarios. Performance results showed that Soilse and SoilseC scale while also providing notably better performance in terms of AWT and AvgStops when compared to best performing baselines. An exception to that was Soilse using Boltzmann in the Trinity scenario. Differences in terms of the number of arrived vehicles from Soilse and SoilseC in the both scenarios against the baselines were generally marginal, however, they were clear against RR20s in the DublinICC scenario. In both scenarios, SoilseC performed better than Soilse in terms of AWT and AvgStops and notably so in the DublinICC scenario were it exhibited a lower relearning cost as well.

Essentially, the use of Soilse and SoilseC has proved to provide an adaptive and responsive optimization scheme for UTC in a decentralized manner. In addition, a better global performance was achieved through collaboration as can be seen from SoilseC's results when compared against Soilse. Both Soilse and SoilseC were responsive by detecting genuine changes in the local traffic pattern that consequently initiate a relative relearning process. Also, Soilse and SoilseC were adaptive by relearning a different policy to optimize for a new traffic pattern.

Chapter 6

Conclusions and Future Work

In this chapter we first summarize contributions of this thesis work and then discuss possible future work.

6.1 Thesis Contribution

This thesis describes a decentralized approach to urban traffic control (UTC) optimization using Reinforcement Learning (RL) and agent collaboration, that is efficient, adaptive and yet responsive to the non-stationary nature of urban traffic.

Chapter (1) provided an overview of the RL and decentralized RL approaches to optimization by interacting with the environment and learning from reinforcements. A historical background for UTC was provided and the challenges that motivated the need for an efficient approach to UTC were presented. The trends in UTC optimization were also discussed including the emerging technologies that can be exploited for better UTC systems such as floating vehicle data (FVD). RL and especially Q-Learning was identified as a promising approach for efficient UTC systems. Our research hypothesis was presented where and argues for the possibility of designing an adaptive and responsive UTC system using decentralized RL and collaboration.

In Chapter (2) we reviewed existing Markov Decision Process (MDP) and RL optimization techniques supporting different learning and action selection strategies. Different elements that contribute to uncertainty in UTC systems such as fluctuations in traffic were also discussed in addition to traffic pattern identification. Three groups of UTC approaches where discussed, i.e., classical systems such as SCATS and SCOOT, non-RL based UTC approaches and RL-based approaches. We identified the possible benefit of a decentralized RL-based UTC optimization approach that is model-free and that

can optimize without an a priori model for traffic as well as being able to cope with the fluctuating nature of urban traffic in an adaptive and responsive manner.

In Chapter (3) we first outlined a set of requirements for efficient RL-based UTC systems resulting from the analysis of existing RL-based UTC systems in Chapter (2). After presenting the motivations behind our design choices, the design of the Soilse approach to UTC including its non-parametric Pattern Change Detection (PCD) mechanism was presented. This covered the type of phases, the relearning strategy used, the reward model used as well as the collaboration specifics in the case of SoilseC. Both the Soilse and SoilseC agent algorithms were detailed as well as the PCD algorithm.

In Chapter (4) the implementation of our design was presented. The generic CRL framework that provides the basic constituents needed for building Soilse and SoilseC agents was presented. In order to build Soilse and SoilseC agents, an agent generator was implemented by customizing and instantiating the CRL framework. The PCD implementation was also described.

In Chapter (5) we described the evaluation of our approach through UTC simulation. The performance of deployments of Soilse and SoilseC using available action selection strategies was compared against the performance of fixed-time UTC, (i.e., round-robin (RR)) and SAT (an algorithm that emulates the behaviour of SCATS) baseline deployments in two scenarios. These scenarios are of a different scale and different traffic patterns were used to represent uniform-low, morning-peak, uniform-high and evening-peak traffic in both scenarios. The relearning behaviour in our approach was also discussed. The performance of a situation where only initial learning occurred (SoilseInit) was compared against the performance of Soilse. In addition, different collaboration modes for SoilseC deployments were evaluated. The performance of SoilseC deployments were also compared against the performance of Soilse deployments in both scenarios.

Performance results show that deployments of Soilse and SoilseC using ϵ-greedy as an action selection strategy outperformed those using Boltzmann and greedy. In the Trinity scenario, Soilse using ϵ-greedy outperformed both baselines, RR and SAT, in terms of Average Waiting Time (AWT) by $\sim -10.42\%$ and $\sim -19.53\%$ respectively. On the other hand, SoilseC provided slightly better performance (compared to Soilse) in terms of AWT against RR and SAT in the Trinity scenario by providing $\sim -11.81\%$ and $\sim -20.78\%$ lower AWT respectively. Soilse and SoilseC using ϵ-greedy have also outperformed the baselines in terms of average number of stops (AvgStops), however, SoilseC provided better performance than Soilse against the baselines in that case, i.e., $\sim -18.20\%$ and $\sim -20.20\%$ less AvgStops against RR and SAT respectively.

In the larger-scale scenario, Soilse and SoilseC using ϵ-greedy and Boltzmann deployments have proved to scale and provide a notably better performance in terms of AWT and AvgStops against

the baselines deployments. A single exception was in the case of Soilse using Boltzmann against SAT_2_1.5 in terms of AvgStops. Remarkably, SoilseC using ϵ-greedy has outperformed RR20s and SAT_2_1.5 by $\sim -91.53\%$ and $\sim -54.56\%$ in terms of AvgStops respectively and by $\sim -89.80\%$ and $\sim -43.27\%$ in terms of AWT respectively. In addition, Soilse using ϵ-greedy also outperformed SoilseInit by providing $\sim -44.75\%$ and $\sim -56.50\%$ lower AWT and less AvgStops respectively.

The performance of Soilse and SoilseC deployments in the larger-scale scenario was clearly better than their performance in the Trinity scenario against the baselines. In addition, ϵ-greedy as an action selection strategy showed better performance against other action selection strategies as it relearns in a manner that does not rely on the policy model in contrast to Boltzmann. Moreover, SoilseC deployments, through collaboration, resulted in better performance against the non-collaborative Soilse deployments in the larger-scale scenario compared to its performance in the Trinity scenario. SoilseC agents showed lower ratios of relearning time to the simulation time compared to Soilse agents in both scenarios. For example, these ratios for the most-affected Soilse agents using ϵ-greedy ranged from $\sim 46\%$ to $\sim 53\%$ compared to only $\sim 19\%$ to $\sim 41\%$ for the most-affected SoilseC agents using ϵ-greedy in the larger-scale scenario. The performance of SoilseC depends on the collaboration mode used and on the scale of the scenario.

The above performance analysis shows that the Soilse approach can potentially provide an efficient decentralized RL-based UTC optimization approach that is adaptive and responsive to the non-stationary nature of urban traffic. Moreover, certain limitations have arisen in terms of evaluation given the nature of the simulator used. The tools available in the UTC simulator did not support variability in different input traffic data for a given traffic pattern. Hence, uncertainty on that level could not be measured. The initialization process of PCD parameters, being dependant on small scale preliminary experiments, could also be seen as a limitation however, it is a one-off effort.

6.2 Future Work

During the design and evaluation of our approach and after analysing the state of the art relevant to our approach a number of possible areas of future work were identified.

In the PCD mechanism used by Soilse and SoilseC agents, a fixed thresholding technique was used for all agents in a given deployment. It could be interesting to explore other thresholding techniques and evaluate their effect on the overall performance, for example, dynamic thresholding techniques where the value of the threshold used can change per agent. For example, this dynamic threshold can change based on the performance of the agent and the previous history of genuine traffic pattern

changes. A model for calculating such a threshold is needed. In addition, a collaborative thresholding technique might be designed where a group of Soilse or SoilseC agents can negotiate a given value for the threshold to be used. Moreover, an automatic (possible collaborative) tuning technique that controls the sensitivity of the PCD mechanism per agent could be investigated.

Given that SoilseC agents all collaborate using a given collaboration mode and frequency as well as use a fixed discount rate on the exchanged rewards, a possibility arises to investigate the potentials for changing these parameters dynamically. This could be through a consensus-based protocol among one-hop neighbours or possibly on a regional level. In addition, a metric that combines the performance and the degree of traffic pattern change of involved agents can then be used for group reparameterization.

Soilse and SoilseC could be extended to deal with multi-objective optimization (also known as multi-policy optimization) where a group of different priority objectives need to be optimized simultaneously for. For example, prioritizing public transport or emergency traffic such as ambulances while controlling usual urban traffic simultaneously. Also, Soilse and SoilseC could be generalized in order to deal with other control optimization problems that has a non-stationary environment.

The Soilse approach should possibly support a certain level of fault-tolerance, especially, in a SoilseC deployment where SoilseC agents might fail. In such a case, collaboration could be adversely affected and hence the performance as well. The sensor information reliability could also be addressed.

Bibliography

B. Abdulhai & P. Pringle (2003). 'Autonomous Multiagent Reinforcement Learning - 5GC Urban Traffic Control'. In *Annual Transportation Research Board Meeting*, pp. –.

B. Abdulhai, et al. (2003). 'Reinforcement Learning for True Adaptive Traffic Signal Control'. In *ASCE Journal of Transportation Engineering*, vol. 129(3), pp. 278–284.

O. Abul, et al. (2000). 'Multiagent reinforcement learning using function approximation'. *IEEE Transactions on Systems, Man, and Cybernetics, Part C* **30**(4):485–497.

M. N. Ahmadabadi & M. Asadpour (2002). 'Expertness based cooperative Q-learning'. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* **32**(1):66–76.

N. M. Ahmadabadi, et al. (2001). 'Cooperative Q-learning: The Knowledge Sharing Issue'. *Journal of Advanced Robotics* **15**(8):815–832.

J. S. Albus (1975). 'A New Approach to Manipulator Control: the Cerebellar Model Articulation Controller (CMAC'. *Journal of Dynamic Systems, Measurement, and Control* **97**:220–227.

F. Bai & H. Krishnan (2006). 'Reliability Analysis of DSRC Wireless Communication for Vehicle Safety Applications'. *ITSC'06. IEEE* pp. 355–362.

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

A. L. C. Bazzan (2004). 'A Distributed Approach for Coordination of Traffic Signal Agents'. *AAMAS* **10**(1):131–164.

J. O. Berger (1993). *Statistical Decision Theory and Bayesian Analysis*. Springer.

F. Bernhard (2002). 'Adaptive signal control: an overview'. In *13th Mini Euro Conference - Handling Uncertainty in the Analysis of Traffic and Transportation systems*, Bari.

C. Bielefeldt, et al. (2001). 'TUC and the SMART NETS project'. In *Proceedings of the IEEE ITS 2001*, pp. 55–60.

M. Bielli, et al. (1994). *Artificial intelligence applications to traffic engineering.* No. 90-6764-171-5ISBN : 90-6764-171-5.

F. Busch & G. Kruse (2001). 'MOTION for SITRAFFIC - a modern approach to urban traffic control'. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pp. 61–64.

L. Buşoniu, et al. (2008). 'A comprehensive survey of multi-agent reinforcement learning'. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* **38**(2):156–172.

C. Cai, et al. (2009). 'Adaptive traffic signal control using approximate dynamic programming'. *Transportation Research Part C: Emerging Technologies* **17**(5):456 – 474. Artificial Intelligence in Transportation Analysis: Approaches, Methods, and Applications.

E. Camponogara & K. J. Werner (2003). 'Distributed Learning Agents in Urban Traffic Control.'. In *Lecture Notes in Computer Science*, vol. 2902, pp. 324–335. Springer.

Y. J. Cao, et al. (1999). 'Design of a Traffic Junction Controller Using Classifier Systems and Fuzzy Logic'. In *Proceedings of the 6th International Conference on Computational Intelligence, Theory and Applications*, pp. 342–353, London, UK. Springer-Verlag.

Y. Chen, et al. (2007). 'A New Method For Urban Traffic State Estimation Based On Vehicle Tracking Algorithm'. In *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*, pp. 1097–1101.

M. C. Choy, et al. (2003). 'Cooperative, hybrid agent architecture for real-time traffic signal control'. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* **33**(5):597–607.

C. Claus & C. Boutilier (1997). 'The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems'. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 746–752. AAAI Press.

Commonwealth of Australia (2009). 'Australia's Digital Economy: Future Directions'.

D. Cosgrove & D. Gargett (2007). 'Estimating urban traffic and congestion cost trends for Australian cities' pp. –.

B. De Schutter (1999). 'Optimal traffic light control for a single intersection'. In *Proc. American Control Conference the 1999*, vol. 3, pp. 2195–2199 vol.3.

A. Di Febbraro, et al. (2004). 'Urban traffic control structure based on hybrid Petri nets'. *Intelligent Transportation Systems, IEEE Transactions on* **5**(4):224–237.

C. Diakaki, et al. (February 2002). 'A multivariable regulator approach to traffic-responsive network-wide signal control'. *Control Engineering Practice* **10**:183–195.

V. Dinopoulou, et al. (2006). 'Applications of the urban traffic control strategy TUC.'. *European Journal of Operational Research* **175**(3):1652–1665.

J. Dowling, et al. (2006). 'Building autonomic systems using collaborative reinforcement learning'. *Knowl. Eng. Rev.* **21**(3):231–238.

K. Dresner & P. Stone (2004). 'Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism'. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 530–537, Washington, DC, USA. IEEE Computer Society.

K. Dresner & P. Stone (2005). 'Multiagent traffic management: an improved intersection control mechanism'. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pp. 471–477, New York, NY, USA. ACM.

K. Dresner & P. Stone (2006). 'Multiagent Traffic Management: Opportunities for Multiagent Learning'.

Dublin Transportation Office (2008). 'Road Users Monitoring Report'.

I. Dusparic & V. Cahill (2009a). 'Distributed W-Learning: Multi-Policy Optimization in Self-Organizing Systems'.

I. Dusparic & V. Cahill (2009b). 'Using Reinforcement Learning for Multi-policy Optimization in Decentralized Autonomic Systems - An Experimental Evaluation'.

S. Eichler (2007). 'Performance Evaluation of the IEEE 802.11p WAVE Communication Standard'. In *Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th*, pp. 2199–2203.

European Commission (2001). 'European transport policy for 2010 : time to decide'. pp. –, Brussels, Belgium.

European Commission (2003). 'Information and Communications Technologies for Safe and Intelligent Vehicles'.

European Commission (2007a). 'Green Paper - Towards a new culture for urban mobility SEC(2007) 1209' **COM/2007/0551 final**:–.

European Commission (2007b). 'Memo - Towards a new culture for urban mobility'.

J. L. Farges, et al. (1983). 'The PRODYN real-time traffic algorithm'. In *Proceedings of the IEE International Conference on Road Traffic Signalling*, pp. 307–312.

G. Felici, et al. (2006). 'A logic programming based approach for on-line traffic control'. *Transportation Research Part C: Emerging Technologies* **14**(3):175 – 189.

R. Fullér (2000). *Introduction to Neuro-Fuzzy Systems*. Advances in Soft Computing Series. Springer-Verlag, Berlin/Heildelberg. ISBN 3-7908-1256-0.

N. Gartner (Transp. Res. Record 906, 1983). 'OPAC: A demand-responsive strategy for traffic signal control'. *U.S. Dept. Transportation* pp. –.

D. C. Gazis (1971). 'Traffic control: From hand signals to computers'. *Proceedings of the IEEE* **59**(7):1090–1099.

C. V. Goldman & S. Zilberstein (2004). 'Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis'. *J. Artif. Intell. Res. (JAIR)* **22**:143–174.

V. Gradinescu, et al. (2007). 'Adaptive Traffic Lights Using Car-to-Car Communication'. In *Proc. VTC2007-Spring Vehicular Technology Conference IEEE 65th*, pp. 21–25.

C. Guestrin, et al. (2002). 'Coordinated Reinforcement Learning'. In *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 227–234, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

M. P. Head, K.L. & D. Sheppard (1992). 'Hierarchical framework for real-time traffic control'. *Transportation Research Record 1360* pp. 82–88.

R. Hoar, et al. (2002). 'Evolutionary swarm traffic: if ant roads had traffic lights'. In *Proc. Congress on Evolutionary Computation CEC '02*, vol. 2, pp. 1910–1915.

P. Hoen, et al. (2006). 'An Overview of Cooperative and Competitive Multiagent Learning'.

S. P. Hoogendoorn & P. H. L. Bovy (2001). 'State-of-the-art of vehicular traffic flow modelling'. vol. 215, pp. 283–303.

J. Hu & M. P. Wellman (1998). 'Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm'. In *In Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 242–250. Morgan Kaufmann.

P. B. Hunt, et al. (1982). 'The SCOOT on-line traffic signal optimization technique'. *Traffic Eng. Control* **23**:190–192.

M. M. Ishaque & R. B. Noland (2006). 'Making roads safe for pedestrians or keeping them out of the way?: An historical perspective on pedestrian policies in Britain'. *The Journal of Transport History* **27**:115–137.

C. Jacob & B. Abdulhai (2005). 'Integrated traffic corridor control using machine learning'. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, vol. 4, pp. 3460–3465.

A. M. M. R. Jeff Schneider, Weng-Keen Wong (1999). 'Distributed Value Functions'. In *Proceedings of the 16th International Conference on Machine Learning*, pp. 371–378. Morgan Kaufmann, San Francisco, CA.

L. P. Kaelbling, et al. (1998). 'Planning and acting in partially observable stochastic domains'. *Artificial Intelligence* **101**(1-2):99 – 134.

L. P. Kaelbling, et al. (1996). 'Reinforcement Learning: A Survey'. *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH* **4**:237–285.

S. Kamran & O. Haas (2007). 'A Multilevel Traffic Incidents Detection Approach: Identifying Traffic Patterns and Vehicle Behaviours using real-time GPS data'. In *Intelligent Vehicles Symposium, 2007 IEEE*, pp. 912–917.

R. v. Katwijk (2008). *Multi-Agent Look-Ahead Traffic-Adaptive Control*. Ph.D. thesis, TRAIL Research School, the Netherlands. T2008/3.

K. P. Katwijk R. (October 2002). 'Coordination of traffic management instruments using agent technology'. *Transportation Research Part C: Emerging Technologies* **10**:455–471.

B. Kerner, et al. (2005). 'Traffic state detection with floating car data in road networks'. In *Proc. IEEE Intelligent Transportation Systems*, pp. 44–49.

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

J. R. Kok & N. Vlassis (2004). 'Sparse cooperative Q-learning'. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, p. 61, New York, NY, USA. ACM.

J. R. Kok & N. Vlassis (2006). 'Collaborative Multiagent Reinforcement Learning by Payoff Propagation'. *J. Mach. Learn. Res.* **7**:1789–1828.

R. Kuhne (2003). 'Potential of remote sensing for traffic applications'. In *Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE*, vol. 1, pp. 745–749 vol.1.

G. C. I. Lin & S. V. Nagalingam (2000). *CIM justification and optimisation.* Taylor & Francis, London, UK.

S.-M. Lin (1999). *Formulation and Evaluation of a Methodology for Network-wide Signal Optimization.* Ph.D. thesis, University of Florida.

M. L. Littman (1994). 'Markov Games as a Framework for Multi-Agent Reinforcement Learning'. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pp. 157–163. Morgan Kaufmann.

H. K. Lo, et al. (2001). 'Dynamic network traffic control'. *Transportation Research Part A: Policy and Practice* **35**(8):721 – 744.

P. R. Lowrie (1982). 'SCATS: The Sydney co-ordinated adaptive traffic system-principles, methodology, algorithms'. In *Proceedings of the IEE International Conference on Road Traffic Signalling*, pp. 67–70.

J. Luo & J.-P. Hubaux (2004). 'A Survey of Inter-Vehicle Communication'.

K. H. . B. F. Matschke, I. (2004). 'Data fusion technique in the context of traffic state estimation'. In *Proceedings of the Triennial Symposium on Transportation Analysis TRISTAN V*, Le Gosier, Guadeloupe.

. D. T. C. Mauro, V. (1990). 'UTOPIA'. *Control Computers Communications in Transportation* pp. 245–252. Pergamon Press, Oxford.

S. Messelodi, et al. (2009). 'Intelligent extended floating car data collection'. *Expert Syst. Appl.* **36**(3):4213–4227.

S. Mikami & Y. Kakazu (1994). 'Genetic Reinforcement Learning for Cooperative Traffic Signal Control.'. In *ICEC*, pp. 223–228.

A. Miller (1963). 'A computer control system for traffic network'. In *Proc., 2nd Int. Symp. on Theory of Road Traffic Flow*, pp. 201–220, London.

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

M. Mitchell (1998). *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*. The MIT Press.

K. Miyazaki & S. Kobayashi (1999). 'Rationality of Reward Sharing in Multi-agent Reinforcement Learning'. In *PRIMA '99: Proceedings of the Second Pacific Rim International Workshop on Multi-Agents*, pp. 111–125, London, UK. Springer-Verlag.

E. A. Mueller (1970). 'Aspects of the history of traffic signals'. *Vehicular Technology, IEEE Transactions on* **19**(1):6–17.

K. Murphy (1999). 'A Survey of POMDP solution techniques'. In *Technical Report*.

K. Nagel & M. Schreckenberg (1992). 'A cellular automaton model for freeway traffic'. *Journal de Physique I* (115):2221.

OECD (2008). *OECD Factbook 2008: Economic, Environmental and Social Statistics*.

K. J. Oh, et al. (2005). 'Variance change point detection via artificial neural networks for data separation'. *Neurocomputing* **68**:239–250.

D. d. Oliveira, et al. (2006). 'Reinforcement Learning based Control of Traffic Lights in Non-stationary Environments: A Case Study in a Microscopic Simulator'. In *CEUR Workshop Proceedings*, vol. 223, pp. –. CEUR-WS.org.

J.-P. Ortega & V. Planas-Bielsa (2004). 'Dynamics on Leibniz manifolds'. *Journal of Geometry and Physics* **52**(1):1 – 27.

L. Panait & S. Luke (2005). 'Cooperative Multi-Agent Learning: The State of the Art'. *AAMAS* **11**(3):387–434.

M. Papageorgiou, et al. (2003). 'Review of Road Traffic Control Strategies'. In *Proceedings of the IEEE*, vol. 91, pp. 2043–2067.

M. D. Pendrith (2000). 'Distributed reinforcement learning for a traffic engineering application'. In *AGENTS'00*, pp. 404–411, New York, NY, USA. ACM Press.

J. Peters, et al. (2005). 'Natural Actor-Critic.'. In *ECML*, pp. 280–291.

I. Porche & S. Lafortune (1997). 'Dynamic traffic control: Decentralized and coordinated methods'. In *Proceedings of the IEEE Conference on ITS*, pp. –.

H. Prothmann, et al. (2008). 'Organic Control of Traffic Lights'.

K. Ramm & V. Schwieger (2007). 'Mobile positioning for traffic state acquisition'. *J. Locat. Based Serv.* **1**(2):133–144.

V. Reynolds, et al. (2006). 'Requirements for an ubiquitous computing simulation and emulation environment'. In *InterSense'06*, pp. 1–, New York, NY, USA. ACM.

S. Richter (2006). 'Learning traffic control - towards practical traffic control using policy gradients - Diplomarbeit'. pp. –. Albert-Ludwigs-Universität Freiburg.

S. Richter, et al. (2007). 'Natural Actor-Critic for Road Traffic Optimisation'. In *Advances in Neural Information Processing Systems*, vol. 19, pp. –. The MIT Press, Cambridge, MA.

D. Robertson (1969). 'TRANSYT method for area traffic control'. *Traffic Eng. Control* **10**:276–281.

D. I. Robertson & R. D. Bretherton (1991). 'Optimizing networks of traffic signals in real time-the SCOOT method'. *Vehicular Technology, IEEE Transactions on* **40**(1):11–15.

F. Rochner, et al. (2006). 'An Organic Architecture for Traffic Light Controllers'. In C. Hochberger & R. Liskowsky (eds.), *GI Jahrestagung (1)*, vol. 93 of *LNI*, pp. 120–127. GI.

A. Salkham, et al. (2008). 'A Collaborative Reinforcement Learning Approach to Urban Traffic Control Optimization'. *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on* **2**:560–566.

M. Satyanarayanan (2003). 'Coping with Uncertainty'. *IEEE Pervasive Computing* **02**(3):2–.

D. Schrank & T. Lomax (2009). '2009 Annual Urban Mobility Report'.

M. Schumacher (1984). 'Two-Sample Tests of Cramer–von Mises- and Kolmogorov–Smirnov-Type for Randomly Censored Data'. *International Statistical Review / Revue Internationale de Statistique* **52**(3):263–281.

S. Sen & L. K. Head (1997). 'Controlled Optimization of Phases at an Intersection'. *Transportation Science* **31**(1):5–17.

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

V. A. Siris & F. Papagalou (2006). 'Application of anomaly detection algorithms for detecting SYN flooding attacks'. *Computer Communications* **29**(9):1433 – 1442. ICON 2004 - 12th IEEE International Conference on Network 2004.

J. C. Spall (2003). *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Hoboken, NJ: Wiley,.

D. Srinivasan & M. C. Choy (2006). 'Cooperative multi-agent system for coordinated traffic signal control'. *IEE Proceedings - Intelligent Transport Systems* **153**(1):41–50.

D. Srinivasan, et al. (2006). 'Neural Networks for Real-Time Traffic Signal Control'. *Intelligent Transportation Systems, IEEE Transactions on* **7**(3):261–272.

M. Steingröver, et al. (2005). 'Reinforcement Learning of Traffic Light Controllers Adapting to Traffic Congestion.'. In *BNAIC*, pp. 216–223.

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

K. Takadama & H. Fujita (2005). 'Toward Guidelines for Modeling Learning Agents in Multiagent-Based Simulation: Implications from Q-Learning and Sarsa Agents'.

M. Tan (1998). 'Multi-agent reinforcement learning: independent vs. cooperative agents'. In *Readings in agents*, pp. 487–494, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

G. Tesauro (2003). 'Extending Q-Learning to General Adaptive Multi-Agent Systems'. In S. Thrun, L. K. Saul, & B. Schölkopf (eds.), *NIPS*. MIT Press.

M. Thottan & C. Ji (2003). 'Anomaly detection in IP networks'. *IEEE TRANSACTIONS ON SIGNAL PROCESSING* **51**(8):2191–2204.

S. Tomforde, et al. (2008). 'Decentralised Progressive Signal Systems for Organic Traffic Control'. In *Proc. Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems SASO '08*, pp. 413–422.

United States DOT (2007). 'Intelligent Transportation Systems for Traffic Signal Control - Deployment Benefits and Lessons Learned'.

United States FHWA (2001). 'Managing Our Congested Streets and Highways'.

United States FHWA (2008). 'SIGNAL TIMING UNDER SATURATED CONDITIONS'. *FHWA-HOP-09-008* .

R. Venkatanarayana, et al. (2007). 'Quantum-frequency algorithm for automated identification of traffic paterns'. *Transportation research record.* (2024):8–17.

J. Visser & L. Molenkamp (2004). 'Vulnerability quick scan of a national road network'. In *The Second International Symposium on Transportation Network Reliability (INSTR)*, Christchurch, New Zealand.

F. Viti (2006). 'The dynamics and the uncertainty of delays at signals' .

H. Wang, et al. (2002). 'Detecting SYN Flooding Attacks'. In *In Proceedings of the IEEE Infocom*, pp. 1530–1539. IEEE.

C. Watkins (1989). 'Learning from delayed rewards'. In *PhD Thesis University of Cambridge, England.*

C. J. C. H. Watkins & P. Dayan (1992). 'Q-Learning'. *Machine Learning* **8**(3-4):279–292.

M. Wiering (2000). 'Multi-Agent Reinforcement Learning for Traffic Light Control'. In *Proceedings of the 17th ICML*, pp. 1151–1158. Morgan Kaufmann, San Francisco, CA.

M. A. Wiering, et al. (2004). 'Intelligent Traffic Light Control' pp. –.

B. Williams (2004). 'CALM Handbook v1.2'.

B. Wolshon & W. C. Taylor (1999). 'Analysis of intersection delay under real-time adaptive signal control'. *Transportation Research Part C: Emerging Technologies* **7**(1):53 – 72.

E. Yang & D. Gu (2005). 'A Survey on Multiagent Reinforcement Learning Towards Multi-Robot Systems'. In *CIG*. IEEE.

L. Yang & F.-Y. Wang (2007). 'Driving into Intelligent Spaces with Pervasive Communications' **22**(1):12–15.

Z.-s. Yang, et al. (2005). 'Intelligent cooperation control of urban traffic networks'. In *Proceedings of 2005 ICMLC*, vol. 3, pp. 1482–1486.