Department of Electronics

MEng Project Report
2015/2016

Student Name: Jacob Busfield

Project Title: Developing Epigenetic Networks to Optimise Traffic Control Systems.

Supervisors: Dr. Martin Trefzer and Dr. Alexander Turner

Department of Electronics
University of York
Heslington
York
YO10 5DD

Developing Epigenetic Networks to Optimise Traffic Control Systems.

Submitted in accordance to the requirements of the University of York for the degree of Master of Engineering.

Jacob Busfield

May 11, 2016

UNIVERSITY of York

Abstract

Traffic congestion continues to rise costing local economies substantially. Where road capacity cannot be physically increased, through new road infrastructure, implementation of more intelligent, adaptive traffic control schemes exists as the likely alternative. However, current versions of these controllers tend to be centralised and thus expensive, unreliable and difficult to scale. Traffic controllers which seek to mimic biological structures, and hence capture their advantages of adaptability, scalability and fault-tolerance, have been previously attempted but with limited success.

This report details epiNet, a computational model able to perform dynamic topological changes, and investigates its effectiveness when implemented as a decentralised, adaptive traffic controller. This is attempted through construction of an agent-based simulation to assess the controller's performance, and evolutionary algorithms which iteratively adapt them to specific, simulated traffic environments.

A methodology to design an epiNet-based traffic controller for a single junction is successfully designed with examples capably handling unseen traffic patterns and junctions with fewer roads. Environments with multiple junctions are also investigated. Evolving duplicate controllers across a symmetrical road design is found to be remarkably effective with the emergence of complex behaviour. Asymmetrical road designs are considered but no effective strategy was found. There are promising avenues of research discussed in this regard.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my supervisors, Dr. Martin Trefzer and Dr. Alexander Turner, for their patience and enthusiasm, all the enthralling discussions, and their insights and guidance throughout this project.

I would also like to thank my family for their endless support.

Declaration of Authorship

I, Jacob Busfield, declare that this thesis titled, 'Developing Epigenetic Networks to Optimise Traffic Control Systems', and the work presented in it, is original except where otherwise indicated. I am aware of the University of York's regulations concerning plagiarism and any work or research which has contributed to this project has been appropriately acknowledged and referenced at their point of use. This includes work that has been previously submitted as a component of a qualification at the University of York or any other institution.

It must be declared that some work presented in this report (specifically some aspects of chapters 1, 2, 4, 6 and project management) has been previously submitted for the degree of Electronic Engineering at the University of York. This was in the form of a report for a module entitled: *Project Literature and Preparation* (ELE00002M).

Hypothesis

This research is motivated by both the notion that epiNet, an artificial epigenetic network, has limited performance in networked applications and that biologically-inspired, adaptive traffic control can reduce congestion. Specifically it is asserted that:

1. Applying a genetic algorithm to evolve a single instance of epiNet in a network limits the performance of the program compared to using a method that allows individuals to apply evolutionary pressure upon one another.

2. Implementing epiNet as an adaptive traffic controller reduces congestion within a traffic network as it inherits useful biological traits including robustness, adaptability and fault tolerance.

Chapter 1

Introduction

Contents

Living organisms, evolving over billions of years, have developed heavily-refined strategies to store, manipulate and process data. In this sense, they can be considered as biological computers. Unlike traditional computers, which tend only to be adept with regard to efficient data processing, emergent properties have developed providing greater robustness, fault tolerance and adaptability. These biological systems therefore provide ample inspiration for designing novel computational artifacts adept in handling unknown, or dynamic, environments. Capturing these traits is a field referred to as biologically-inspired computation which has become increasingly indispensable in many engineering disciplines. The work presented in this report considers one such program, epiNet, and seeks to research new ways in which further biological mechanisms may be applied to improve its performance.

1.1 Motivation

1.1.1 Research

EpiNet, an artificial epigenetic network, shows promise of becoming prominent within the field of bio-inspired computation owing to its capability of displaying complex, dynamic behaviour. It has already been found effective in many applications including audio and visual recognition, data analysis and control engineering [1][2].

An evolutionary algorithm, which replicates sexual selection in silico, can be used to evolve epiNet and optimise a problem's solution. For instance epiNet was able to solve the coupled inverted pendulum task, where it must control a cart and swing connected pendulums to be balanced vertically, by using such a method [2]. However, research conducted into the way a network of epiNet controllers should be evolved is limited. In nature there exists various methodologies where individuals put evolutionary pressure upon one another, or maintain shared genetic information while behaving differently. Many of these processes have successfully been captured artificially often to great benefit to the network they are applied to. These processes however have never been applied to epiNet.

This projects therefore seeks to investigate the effectiveness of using different evolutionary techniques to evolve a network of epiNet controllers. It is hoped that these investigations lead to controllers consistently evolving more complex behaviour than they would otherwise.

1.1.2 Tackling Congestion

As the world's population continues to rise so does the capacity requirement placed upon our road networks. INRIX, a leading traffic forecaster, estimates by 2030 that the annual cost of congestion in the United Kingdom will rise 63 percent to £21.4 billion[3]. Local councils are often helpless to prevent this with new road infrastructure owing to their unjustifiable expense or by spacial restrictions. Where investment has been unavoidable huge centralised systems, such as New York's $1.6 million system[4], have been built. These system are inherently flawed in terms of security, maintenance costs and low fault-tolerance[5]. There is therefore a pervasive need for a modern alternative.

Current systems, such as the internet, demonstrate that decentralised computation is overwhelmingly the best conceivable approach that exists. It removes the ability of attackers to

gain control of the entire system by compromising just one access point, and allows significantly improved scalability as there is no requirement for elaborate fully-connected networks[5]. Bio-inspired computational methods could also be applied to further augment traffic control performance. Research shows such systems: require much less initial calibration massively reducing the cost of installation; are able to handle isolated faults by dynamically changing their social behaviour; and can be evolved to find optimal controller configurations even when facing multi-objective problems[6]. These approaches have been previously considered, and while simulations showed promising results, developing controllers with enough instructions to exhibit adaptive behaviour led to overfitting[7][8].

This project therefore considers artificial epigenetic structures, namely epiNet, and how traffic controllers could replicate their ability to allow environmental factors, and their current state, to physically alter how the controller interprets its instruction set in order to prevent overfitting. From this it is hoped, via the use of agent-based simulations and different evolutionary techniques, that an adaptive, decentralised controller may be realised.

1.2 Project Outline

The main aim of this project is to investigate how epiNet should handle a network of controllers. A traffic control application facilitates this research by providing an ideal, observable environment. The secondary aim is to determine the effectiveness of epiNet, as an adaptive traffic controller, in reducing congestion. The secondary aim seeks only to ascertain feasibility, as developing highly realistic models for more conclusive assessment would detract from the main research goal.

The project is split into two phases. The first seeks to construct the required software components: a reduced traffic simulation to simplistically model traffic; an epiNet controller that is integrated into the simulation; and an evolutionary algorithm that optimises controllers. The second phase, by building upon these foundations, then seeks to investigate the project's research aims by exploring different methodologies that use epiNet to control multiple traffic controllers.

1.3 Contributions

The work within this project has made the following contributions to knowledge:

- Realisation that the averaging of fitness tests within an evolutionary algorithm, which looks to optimise an epiNet-based traffic controller within an agent-based simulation, should be conducted using the mean.
- Realisation that evolving duplicate epiNet-based traffic controllers on a symmetrical traffic network can lead to the emergence of complex behaviour.
- Realisation that a single instance of epiNet should not be mapped to multiple controllers as the resulting inputs and outputs are complex and can become too difficult to evolve.
- Realisation that evolving multiple separate controllers simultaneously within a traffic network is often too computationally expensive to be viable.
- Realisation that asymmetrical traffic networks, or ones with junctions with inconsistent road numbers, are best handled by epiNet when the constituent controllers are separately evolved on separate, purpose-built simulations.

1.4 Report Structure

This report has a general structure of three parts. The first (chapters 2 - 6) introduces epiNet, an agent-based traffic simulation and evolutionary algorithms with supporting chapters providing pre-requisite knowledge where necessary. The second part (chapters 7 and 8) then expand upon these foundational processes to present a set of investigations conducted into the research area. Finally chapter 9 concludes the report, summarising key findings and outlining possible future research avenues. A consideration of project management is also given. More specifically:

Chapter 2 By first introducing basic biological principles, defines epigenetics and presents the epigenetic mechanism epiNet was based on.

Chapter 3 Describes epiNet, its structure and how, and why, it was ported to C++.

Chapter 4 Discusses the effectiveness of different traffic control schemes and the pervasive need for a modern alternative.

Chapter 5 Presents the agent-based simulation developed in this project.

Chapter 6 Introduces biological evolution before defining evolutionary algorithms.

Chapter 7 Investigates how a methodology to evolve a single epiNet-based traffic controller for agent-based simulation was developed.

Chapter 8 Investigates how effective different evolutionary schemes are in developing networked traffic control using epiNet.

Chapter 9 Summarises the work presented, drawing conclusions and suggesting future lines of research.

Project Management Outlines the methodology used and evaluates its effectiveness.

Chapter 2

Epigenetics

Contents

This chapter introduces epigenetics such that it may be referred to later within this report. To understand how epigenetic mechanisms function foundational biological principles are first presented. This focuses on: what objects perform tasks within cells; how these objects are created; and ways in which these objects interact. Epigenetics then expands this understanding by detailing how creation of the aforementioned objects can be influenced by a cell's local environment thus changing the cell's behaviour. The biological advantages found for using such a process are also outlined.

2.1 Proteins

This section seeks to describe proteins based upon [9]. Proteins are large biological molecules, or macromolecules, that perform the majority of essential processes within an organism's cell. They vary from $1\mu m$ (such as tintin) down to $< 1nm$ (such as Trp-cage [10]). Due to

this small scale, interactions are atypical; they tend to be dominated by electrostatic charges rather than being physical. Proteins consist purely of amino acids, simple organic compounds, of which there are 20 standard forms (though some rare exceptions exist). These amino acids are in the structure of at least one linear chain, or *polypeptide*, where both its constituent amino acid types (or *residues*) and structure relative to other polypeptides determine the protein's behaviour. This behaviour is highly specialised allowing proteins to perform versatile processes within, as described in [11], one of four sets:

- *Binding* - to other proteins to modify their behaviour in some way.
- *Catalysis* - which inhibits or excites gene expression.
- *Structure* - such as modifying the cell wall.
- *Switching* - or altering of the local environment around the protein.

2.2 Nucleic Acid

This section introduces nucleic acids, namely *DNA* and *RNA*, by taking from work presented in [12] and [13]. Nucleic acids are organic structures which, for all known life, are responsible for storing an organism's genetic information. Nucleic acids are long macromolecules, or *bipolymers*, made up of a set of repeating base components, or *nucleotides*, whose combination determines what information is encoded. This structure allows for efficient access, duplication and manipulation. Deoxyribonucleic acid (DNA) is a nucleic acid responsible for retaining the majority of genetic instructions within an an organism. Its structure, as discovered by Crick & Watson[14], is that of two bipolymer strands where each contains nucleotides, or *bases*, of: guanine(G), cytosine(C), adenine(A), and thymine(T). The nucleotides between strands only ever bond in the pairs AT and CG to form the well-established double helix structure. This structure has been found to be remarkably stable which is crucial as it must retain information, without allowing corruption, over successive generations. Ribonucleic acid (RNA) is similar to DNA though generally contains only a single bipolymer strand, and contains uracil(U) to be the complementary base of A (instead of T). The loss of structural rigidity allows RNA greater malleability compared to DNA which is vital for certain processes (such as protein synthesis discussed later).

2.3 Protein Synthesis

This section details protein synthesis as presented by Tribe[15]. Protein synthesis is the process by which proteins are created (*synthesised*) with reference to a DNA or RNA template. To understand how this is achieved first consider figure 2.1.

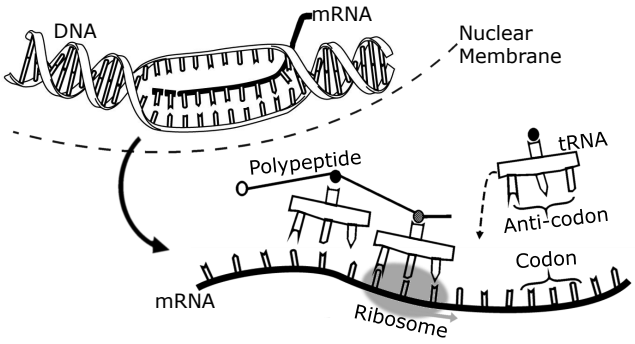

Figure 2.1: Illustration of the key parts required for protein synthesis namely mRNA transcription of DNA and the translation of mRNA by tRNA.

The first stage, known as *transcription*, is the process of copying a region of DNA (or RNA). To achieve this DNA is *unzipped*. This is where the bonds between base pairs, within the section of DNA required, break apart. The mechanics of how, and why, this is achieved is beyond the scope of this report. This step is also not applicable to RNA which only has a single thread (there is nothing to unzip). Messenger RNA (mRNA) can now be formed. This is achieved when RNA nucleotides bond to the newly exposed DNA bases. These nucleotides, now a copy of the underlying DNA code, now join together to form the mRNA. This can then leave the DNA and the cell's nucleus, entering into the cell's cytoplasm, allowing the DNA to zip itself back up (ready for subsequent transcriptions).

The final stage, known as *translation*, is the process of creating a protein from the mRNA. First a complex molecular machine, known as a *ribosome*, straddles the mRNA strand. It reads the mRNA three bases, or *codon*, at a time. Reading must begin on the mRNA at the location of the *start codon* (usually denoted by a base structure of AUG[13]). Each time the codon is read the ribosome uses a specialised form of RNA, known as tranfer RNA (tRNA), which has three nucleotides that are the exact base pairs for the codon (this is known as the anti-codon). This allows the tRNA to bind to the mRNA. The tRNA has a specific amino acid based upon what the structure of its anti-codon is. The ribosome allows each specific amino acid to create a link, or *peptide bond*, with the amino acid from the previous read. In this way a chain of amino acids, or *polypeptide*, is formed. This process is continued until the

stop codon is read. A protein is created by allowing the polypeptide to fold.

2.4 Genes

This section describes genes as presented in [13]. During protein synthesis it was seen how distinct regions of DNA could be used to form specific proteins. These distinct regions are known as *genes*, and when they are used to synthesise a new protein the event is referred to as *gene expression*. A Genes' sequence can develop mutations leading to a new variant of the gene, known as an *allele*. These alleles can cause the observable traits, or *phenotype*, of an organism to change [16]. The entire set of genes for an organism is known as an organism's *genotype*[16]. It should be noted that the definition of the gene given in this paper is simplified from the modern definition which accounts for special cases (such as non-coding genes and split coding regions[17]) as to stay consistent with the computational model later presented in chapter 3.

2.5 Gene Regulation

The section briefly introduces gene regulation, taking work from Latchman [18], before considering the work of Jacob & Monod [19] which presents a simple known example: the lac operon. Gene regulation is the process by which gene expression is controlled to maintain an optimum state. This process is achieved through the interactions of many complex, application-specific structures and sub-processes (leaving detailed inspection outside the scope of this paper). In general, although the rate of gene expression can be varied at most stages of protein synthesis, the dominant control system is found at the point of transcription. This system requires that a specific *regulatory protein* binds with a *regulatory sequence*, a small section of DNA found upstream of the required gene, before the gene can be expressed. The concentration of the regulatory proteins determine the rate of gene expression.

Even the smallest gene regulatory systems have much more complex structures than described. Consider briefly one of the smallest: the lac operon (found in the bacteria: *Escherichia coli*). This system divides the regulatory sequence to also include a section, known as an *operator*, that naturally prevents any of the three genes in this system from being expressed as long as a *repressor* molecule is attached to it. The repressor will only detach when lactose binds to it changing the repressor's form thus allowing the genes to be expressed as normal.

2.6 Biochemical Networks

This section outlines two biochemical networks, fundamental to the understanding of the epiNet model, by taking work predominantly from Bower & Bolouri [6]. Biochemical networks are extremely complex systems, consisting of unconnected biological components, capable of dynamic behaviour and self-organisation. There are two such networks that will be briefly detailed in this section: gene regulatory networks (GRNs) and cell signalling networks (CSNs). GRNs are responsible for high-level emergent behaviour seen within organisms. This arises from the interactions between genes via proteins. These proteins, as discussed in section 2.1, have the ability to: bind to other proteins; inhibit or excite gene expression; modify the cell wall; and even change the local environment [9]. As GRNs massively vary in size, and as gene regulatory systems themselves can be very complex, the mapping of such networks is incredibly difficult. However, this complexity allows GRNs to evolve with great specificity, adaptability and robustness. CSNs act as communication links between a cell and its environment[20]. These networks process these signals and deliver messages that are suitable for the recipient. For example a high salinity environment around the cell will trigger CSNs to alert the GRN responsible for homoeostasis with regard to salinity.

2.7 Epigenetics

This section looks to define epigenetics such that references to it remain consistent. Epigenetics is the study of differences found in an individual's phenotype caused without alteration to their genotype[21]. Turner in [20] argues that there is debate over the exact definition of epigenetics and that for the sake of producing a computational model to declare one as:

- Epigenetics only includes systems which alter gene expression without altering any underlying genetic code.
- Epigenetic mechanisms are reversible.
- Epigenetic modifications are stable.

It should also be noted that while epigenetic systems are often regarded as heritable [21], there exists at least one exception to this and as such heritability can not be included within the definition of epigenetics [20]. This paper will utilise the definition presented as to stay consistent with the computational model seen in chapter 3, as it was also produced by Turner under these assumptions [20].

2.8 Chromatin

This section, based upon work by Craig & Wong [21], introduces chromatin, an epigenetic structure whose behaviour significantly underpins the epiNet model. Chromatin forms one of the major epigenetic structures within a cell nucleus. To understand its function first consider figure 2.2 which shows a single chromatin fibre. In this figure it can be seen that DNA is wrapped around *histone octamers* to form a compact fibre. Histone octamers consist of 8 core *histone proteins* which act as scaffolding to allow the DNA to wrap around them. Histone proteins also have tails, which can move, that determine which genes can be actively transcribed and thus expressed. The exact mechanism of how this is achieved, however, is currently unknown. Note that this paper treats chromatin as an epigenetic structure as to stay consistent with Turner's work on artificial epigenetic networks even though some research contests aspects of this[1][20][2].

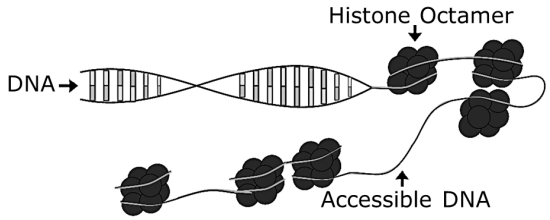

Figure 2.2: Illustration of DNA being wound into a chromatin fibre taken from [1] (note that shading and font has been adapted for consistency).

2.9 Advantages of Epigenetic Mechanisms

This section seeks to detail three main biological advantages of epigenetic mechanisms:

Genetic Packaging

Higher order folding of DNA around histones, to form chromatin, allows it to fit within a nucleus [21]. Without this structure the amount of genetic information an individual could hold would be severely reduced. This is exemplified by eukaryotic, which contains chromatin, which has a genotype and phenotype that is vastly more complex than that of Prokaryotic cells (which contain none) [21].

Genetic Memory

Bonasio et al. in [22] found that epigenetics give a rudimentary form of memory. It was seen in section 2.7 that epigenetic processes are reversible and do not change any underlying genetic code. Previous states can therefore be transitioned between. Transitions made via

chromatin, though dependent on the local environment, can also be influenced by the current state allowing state-machine like structures to form [21][22]. This ultimately leads to much more complex phenotypes being developed than would otherwise form.

Switch-like Mechanic

Epigenetics, by changing which genes are transcribed, allows a large number of individual states to be completely interchangeable from one another. Changes between states, triggered by local environmental changes (which in turn may be a response in themselves to external changes), occur quickly and consistently [21]. This allows cells to radically *switch* their behaviour and thus be more capable at handling a greater array of situations (regardless of how unlikely they are to occur).

2.10 Summary

This chapter detailed how proteins perform the majority of fundamental tasks within a cell. It was also described how proteins are synthesised and ways in which they can react with one another (in biochemical networks for instance). This set the foundations to introduce chromatin, an epigenetic structure, which allows the local environment of a cell to physically change which genes are transcribed and thus the behaviour of the cell as well. This structure was seen to have the following advantages: allows for a more complex genotype as it can be folded to reduce size; a rudimentary form memory that can be used to influence future cell behaviour; and the innate trait of handling dynamic environments well.

The following chapter details how these biological features, mainly chromatin, have been captured artificially.

Chapter 3

EpiNet

Contents

This chapter introduces epiNet which is a computational model that is able to perform dynamic topological changes inspired by the function of chromatin within cells. To achieve this the motivation for such an application is first introduced. The main mechanisms which define the model, as demonstrated by Turner et al. in [1], are then presented by building upon biological concepts introduced in chapter 2. A brief consideration of the effectiveness of epiNet, determined by published research, is then stated. Finally technical detail of how epiNet was ported to C++, and why this work was conducted, is given.

3.1 Introduction

Often a set functions are required to transform a set of inputs to a set of outputs. Consider image classification problems whereby an object within an image needs detecting and the type determining. Such a problem could see each image pixel being a function input and the object classification as the output. This is obviously a challenging problem with functions being too

difficult, and too time-consuming, to explicitly write. Biologically-inspired approaches have therefore been developed which make this design process significantly easier; their approach is heuristic in the sense they seek not to find a perfect, optimal solution, but rather embody adaptable, robust designs found in biology to find a practical method that performs sufficiently well. EpiNet is such an approach that takes inspiration from epigenetics, specifically chromatin, within a cell. It considers inputs as changes in its local environment and, by actively allowing this to affect what parts of its genetic information are expressed, dynamically changes the resultant output. The advantage of such a methodology (as seen in section 2.9) is that, as switching between scenarios or cases is handled structurally, epiNet's constituent processes can remain relatively simple and modular.

3.2 Structure

The structure of epiNet can be understood by first considering its graphical illustration given in figure 3.1. From this it can be seen that the main artificial, genetic structure consists of

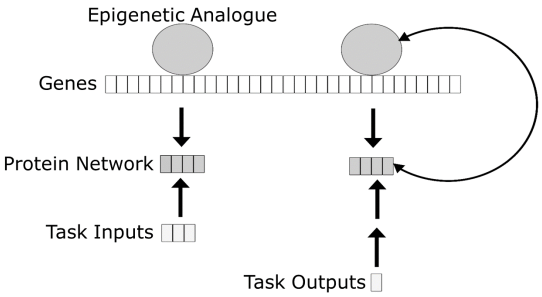

Figure 3.1: Illustration of a single iteration of epiNet. The genes directly below the epigenetic analogue are copied into the protein network where they can interact with input or outputs as specified by each gene. This figure is taken from [1] (note that shading and font have been adapted for consistency).

a singular strand of genes, or *genome*, existing on a linear scale between [0,1]. Straddling the genome exists a fixed-width object, which mimics the functionality of chromatin, that selects which portion of the genome is transcribed and thus which genes are expressed. In this model, each active gene synthesises a single artificial protein. Within a single epigenetic analogue, multiple connections are formed between proteins creating a protein network. The connections between these proteins are pre-determined by the properties of their parent gene in a methodology referred to by Turner (first in [20]) as the reference space. Connections act as either inputs to, or outputs from, either other proteins or the inputs and outputs of the entire epiNet structure. The protein: sums its inputs with pre-determined weights; converts

the result through an internal sigmoid function; before outputting the value through its output connections. The larger, combined network can therefore perform complex mathematical operations between the inputs and outputs of the system. The epigenetic molecule then uses the protein network, and its own internal sigmoid function, to determine where to subsequently move. This allows, as the original protein network has been destroyed, for a new combination of genes to be expressed and thus a new protein network to be created. This in turn, changes the mathematical operation performed between the inputs and outputs of epiNet depending on the local state of the system.

3.2.1 Reference Space

To understand what the reference space is, and how it operates, first consider a single gene within epiNet. This gene, illustrated in figure 3.2, has a fixed width (or *proximity*) and an *identifier* point which can be used to determine the gene's position.

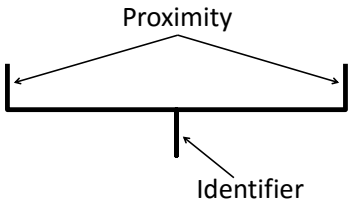

Figure 3.2: Illustration of how a single gene is perceived in the *reference space*. The gene has an identifier which is used to determine the genes position in the reference space. It also has a proximity, a fixed-width bound, that determines the amount of reference space the gene is present in.

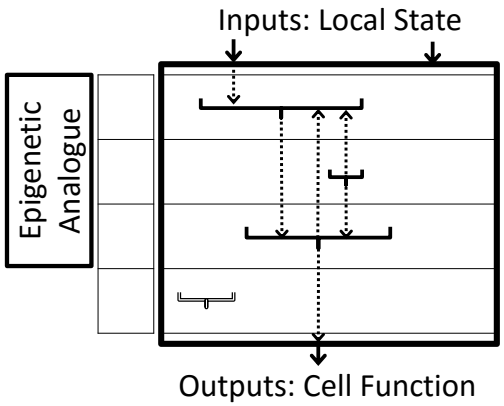

Figure 3.3: Illustration of the reference space. The epigenetic analogue (left) straddles the vertically aligned genome. The analogue activates the three genes it touches, the other gene is therefore inactive. The three genes form connections with one another if their identifier is within the proximity of another gene. Only one input is utilised, and it is connected to the top gene. The output is connected to the bottom gene. This reference space is used to form the protein network and its connections.

Now consider figure 3.3 which shows the reference space. The genome is pictured now vertically on the left-hand side of the diagram with the epigenetic molecule straddling it as before (in section 3.2). Three genes can be seen to be activated by the epigenetic analogue. Between these genes exist connections if a gene's identifier is within the proximity of another. Note that multiple connections can exist for any gene. Connections are also formed between an input, or output, and a gene if they lay within its proximity. The outputs must be mapped to a gene even if no gene is within its proximity (using a priority-based system). Inactive genes do not form connections, and connections are only formed between genes in the same analogue (if there are multiple). Connections can form loops, such as the one seen in the figure 3.3, forming rudimentary internal memory.

3.3 Research

EpiNet has been found to be useful in a wide range of applications. The Artificial Epigenetic Network (AEN), a predecessor of epiNet, was found in [2] to significantly outperform a similar methodology (which did not use epigenetic mechanisms) at a coupled inverted pendulums task (described in [23]). The results showed that over multiple tests, that not only was the mean case improved, but that range of times varied less as well. In [1], epiNet was found to be able to perform edge detection. The source showed that the modular nature of epiNet supported partitioning of its genome for the different types of behaviours required.

3.4 Porting EpiNet to C++

C# Code for the epiNet program was provided by Dr. Alexander Turner (a supervisor of this project and original author of EpiNet [1]). This code in the form presented could not interact directly with the agent-based simulation (later presented in chapter 5) which had already been written in C++. It was therefore decided to port epiNet (the smaller of the two programs) to C++. This section seeks not to detail the entire working of epiNet, which is better presented by [20][2][1], but instead provide: reason why porting epiNet was chosen over integration of both languages; an overview of the code structure; and detail of the differences between both versions. Note that the ported code was tested and verified as working under the close supervision of Dr. Alexander Turner.

3.4.1 Avoiding Dynamic-link Libraries

Using a Dynamic-link library (DLL), as described by [24], was considered and ultimately rejected in this project. A DLL is a library type created by Microsoft that contains code and data which can be used by multiple programs simultaneously. Both C++ and C# code can be packaged into a DLL and both languages can run DLLs. Hence it was considered as a method to allow epiNet (C#) to use (the simulation's) C++ functions. Complete instruction is not given by this report except to say the attribute *declspec* can be used to specify class-storage by using the *dllimport* and *dllexport* commands[1]. Example code for this is given by listing 1 in the appendices. Porting epiNet to C++ was chosen over using DLLs as using DLLs proved challenging to implement (thus expected to take longer than porting to complete) and a ported version of epiNet was preferred as it allows more intuitive integration of the simulation and epiNet modules.

3.4.2 Code Structure

The motivation for design choices given in this section are entirely motivated by a desire to keep the ported code as structurally similar to the original code as possible. The structure of classes that form epiNet is given by figure 3.4.

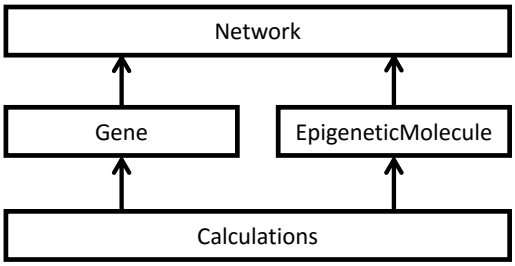

Figure 3.4: Block diagram illustrating how classes interact in the C++ version of epiNet. Square boxes represent classes. Arrows indicate that a class is inherited by the class it points to. The *Network* class is the top level class of epiNet.

It can be seen that the Network class is the top level class of the epiNet program. It stores a vector (a class of sequence containers in linear arrangement defined in C++'s standard library [25]) of Genes to create the genome used by epiNet. Gene is a very simple class holding information associated with that gene, such as weight, offset and so on, and getter and setter methods. Network also stores a vector of EpigeneticMolecules to create a set of epigenetic

[1] `__declspec` resource: `msdn.microsoft.com/en-us/library/3y1sfaz2.aspx` [Accessed: 02/05/2016]

analogues, referred to as the epigenome, used within epiNet. Like Gene, EpigeneticMolecule is a simple class with simple data parameters and getter and setter methods. It also contains an execute function which is used to update and determine the position of the analogue. The length of both the genome and epigenome are determined by parameters passed to Network's constructor. This also sets the number of inputs, outputs and the ratio of both in the reference space. An alternative exists to random initialisation; epiNet designs can be saved to, and loaded from, `.txt` files.

Once an instance of epiNet has been initialised it is ready to use. This is achieved by: assigning values to epiNet's inputs (using `setInputs()`); executing the network (using `executeGeneticNetwork()`); and then retrieving the outputs (using `getOutputs()`). This process can be repeated indefinitely. Execution is more fully described by psuedocode in listing 1.

Algorithm 1 Execution of epiNet

```
1: for Each epigenetic analogue (epigeneticMolecule) do
2:     Determine the position of the epigenetic analogue.
3:     for Each gene do
4:         if Gene is within the proximity of the epigenetic analogue then
5:             Transcribe the gene (make a copy and add to analogue's protein network).
6:     Form connections in protein network (using reference space).
7: for Each protein do
8:     Use connected proteins to determine protein's weighted expression.
9:     Use Sigmoid function to update protein's expressed value.
```

3.4.3 Differences Between Original and Ported Versions

This section assumes the knowledge that the differences between C++ and C# are understood. As such, this section will not describe syntactical or structural changes between both programs. Instead only design choices, or other compromises, which had to be made to accommodate successful porting are presented with justifications given.

Replacing Arrays

Arrays (in C#) are often declared, but not initialised, and later assigned memory using the new command. This could be replicated in C++ using a pointer and then declaring memory but, as C++ doesn't enjoy the same robust memory management as C#, it was decided that using vectors would be safer (with regard to leaking memory). Vectors are also used in place of C#'s list container (being equivalent) and using vectors consistently for all linear memory management seemed to make more intuitive sense.

Removing Static Class Variables

In the original implementation it is required that the user use the methods: `setInputSpace()` and `setOutputSpace()`. These are used to universally set the number, and ratio, of inputs and outputs of networks pre-emptively. This can be easily mimicked in C++ using static class variables. However, the function was changed so that these variables became local to their class and set by the Network's constructor. This allows for more modular, readable code that enjoys greater functionality. For instance epiNet objects of different sizes can be constructed in the same program and instantiation of epiNet can be achieved within a single line of code.

Printing Networks

Printing and reading a network's critical parameters to a `.txt` file is unchanged from the original implementation (the same file is produced) except the `fstream` library[2] is required to open, read and write them. The error of not being able to open the network file is also explicitly handled by printing the error to the console, temporarily sleeping the program (giving time for error to be read) and exiting.

3.5 Summary

This chapter introduced epiNet. It first noted that, by taking inspiration from biology, that programs can be developed that sufficiently solve a problem that would otherwise be elusive. It was seen how chromatin, and how it modifies DNA transcription, was captured artificially to form the general structure of epiNet. It was noted how a C# implementation has been effective in various applications in published findings. A need in the project for a C++ port of epiNet was finally stated with consideration given to its structure and how it differs from the original implementation.

This C++ implementation of epiNet will be referred to later in this report when it is utilised in an agent based simulation to control traffic.

[2] `fstream` tutorial: `www.cplusplus.com/doc/tutorial/files/` [Accessed: 02/05/2016]

Chapter 4

Traffic Control

Contents

This chapter introduces traffic control systems, outlines common examples and emphasises the pervasive need for more efficient designs. Terminology and traffic infrastructure used within this section refers to general British practise as outlined in [26].

4.1 Traffic Control Schemes

Traffic control systems modify signals shown to its users to change their behaviour and thus augment the flow of traffic within a system (whether this be with regard to safety or the speed of traffic flow). Traffic systems must deal with highly specialised traffic environments which will have varying: infrastructures, demands, user types and so on. Traffic systems deal with this by either being incredibly generic (inefficient) or incredibly individualised (expensive).

In the UK traffic control systems are falling short of the mounting requirements asked of them. A study conducted by INRIX[3] found that between 2013 and 2030 that the estimated total cumulative cost of congestion to the UK economy is £307 billion, with the annual cost of congestion set to rise by 63 percent to £21.4 billion over the same period. This is seen as

unavoidable as traffic systems are unlikely to improve and road capacity improvements (via new infrastructure) is limited due to cost (exceeding local budgets) and spacial requirements.

4.1.1 Fixed-time Control

Fixed-time traffic controllers are the simplest controllers that exist. They utilise a simple state machine that transitions between states (such as traffic light signals) based upon timers and logic (that prevent dangerous signal combinations). If each road connection is serviced sequentially the controller is known as a round-robin controller. These systems are rarely seen outside of temporary traffic systems.

4.1.2 Dynamic Control

Dynamic controllers are identical in operation to fixed-time traffic controllers except the jump between states can be influenced by information it receives from non-intrusive traffic-monitoring sensors. These sensors can vary from metal detectors buried under roads to video image processing, and can be used to detect any agent within the system.

4.1.3 Coordinated and Synchronised Control

This section takes work from [27] in its presentation. Modern traffic controllers are born from the philosophy that road-users should encounter as many continuous green traffic lights as possible. Systems therefore look to segment traffic flow into small sections, or *phases*, that they then pass through cascading traffic lights. The envisioned benefits are as follows:

- Increases road capacity and journey times.
- Reduces collisions and driver frustration.
- Reduces pollution, fuel consumption, noise and vehicle wear (exacerbated by repeated starts).
- Drivers conditioned to drive the speed limit as not encounter as many red lights (known as the *green wave*).

This system design is achieved by either centralised computation, known as a *coordinated system*, or by allowing nodes within a system the ability to update neighbouring nodes in a decentralised manner, known as a *synchronised system*. Both these systems, however, tend to

be poor at dealing with changing traffic behaviour, that has not been explicitly prepared for, due to their fixed scheduling design.

4.1.4 Adaptive Control

Adaptive control is the cutting edge of traffic controller design. These systems look to address the non-adaptability shortcomings of coordinated and synchronised traffic controllers. There are two main branches for this: adaptive, fully-centralised, coordinated systems and bio-inspired synchronised systems. Little research has been conducted on the latter with only minor advances showing that extended neural networks and stochatic networks can improve the flow of high volume traffic within restricted simulations[7][8]. Investment therefore turns to the former in the absence of a better solution, leading modern traffic systems to become ever more complex, expensive and individualised. They also inherit the problems of centralised computation such as low fault tolerance, poor scalability and reduced security[5].

4.2 Case Study: Midtown-in-Motion

This section introduces the Midtown in Motion traffic controller as presented by [4]. The *Midtown in Motion* system is a mammoth adaptive coordinated traffic control system implemented within part of the city of New York. It utilises conventional sensors to track the location of around a million vehicles each day. The system runs autonomously, over dedicated carrier-free mobile broadband (NYCWiN), reacting to traffic by a complex set of pre-defined behaviour. Manual operation is available if required. It has shown to reduce travel times by only 10%. The installation of the system cost $1.6 million with maintenance costs expected to be considerable.

4.3 Summary

This chapter presented traffic control systems and outlined four common examples. The most important case from these, with regard to this project and its aim, was adaptive control. It was seen that this cutting-edge design, which allows traffic to affect its behaviour in real-time, is the best hope in tackling growing congestion. However the two branches investigated are both flawed. Centralised systems (such as Midtown-in-motion) are: expensive, have low fault-tolerance, scale poorly, and are innately insecure. Bio-inspired approaches have received

too little research, and the programs used thus far have been poorly suited showing only minor improvements.

The following chapter will detail how a simulation was built to simulate fixed-time control (in the form of a round-robin) and bio-inspired, adaptive control. The report will then later use this to determine if epiNet is, unlike other bio-inspired models, well suited as an adaptive traffic controller.

Chapter 5

Agent-Based Simulation of Traffic Control

Contents

This chapter introduces the agent-based traffic simulation, used to determined the performance of different control systems upon the behaviour of traffic, that was developed in this project. Terminology and design principles implemented are influenced by the work of Barceló in [28]. The chapter first presents: why agent-based simulation was chosen; justifies the complexity of it; and credits the graphical library it was built upon. The high-level functionality is then considered before the code structure is technically described. Finally brief consideration is given of the main problems encountered.

5.1 Introduction

An agent-based traffic simulation models every car (*agent*) as it travels through a virtual road environment and reacts with roads, traffic lights and other cars. These simulations seek to stress the control scheme with both levels and directions of traffic to determine the strengths and weaknesses of it. Agent-based simulation can also be modelled graphically which is useful for visual, real-time inspection of traffic. This allows simulations to be easily debugged, demonstrated and behaviour inferred. These points were salient in choosing it over other alternative simulation strategies (which are not explored in this report). The *Simple and Fast Multimedia Library* (SFML) was used in the development of these graphics to reduce the complexity (and therefore time and risk) of implementation. Models can be as detailed as required, though increased complexity increases the computational time required to execute them. With view of this, a *reduced* traffic simulation which takes a simplistic view of traffic, was used. This allows for increased testing and clearer conclusions to be drawn which is preferential with regard to the research aims of this work. The simulation was programmed in C++ as its a well-documented language, has a good compromise between object-orientation and low level optimisation and supports the SFML library.

5.2 Functionality Overview

The agent-based simulation developed was designed to be intuitive, adaptable and informative. The Graphical User Interface (GUI) for this simulation can be seen in figure 5.1.
There are four main objects present within a simulation:

- *Cars* - these are the singular agent used within the simulation. They have limited functionality as to remain computationally inexpensive. They can: accelerate; drive toward a position; follow a route; and stop at red lights and behind cars.
- *Roads* - these objects exist between two points on a two dimensional surface. The road has two lanes with traffic flowing in opposite directions.
- *Locations* - these nodes spawn or destroy cars at a fixed position within a two dimensional plane. Each connected road is given its own set of traffic lights, and cars will only enter the location when their light is green.
- *Systems* - these nodes, which exist at a fixed position within a two dimensional plane, allow cars to pass from one road connection to another. Each connected road is given

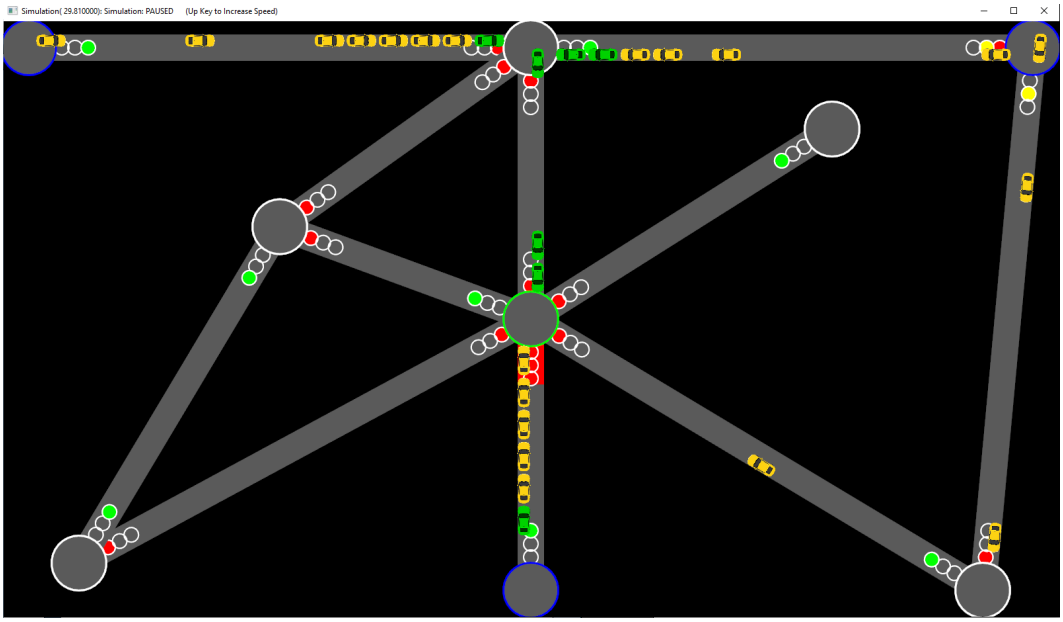

Figure 5.1: Screen capture of the agent-based simulation developed in this project. *Blue* circles are locations that spawn and remove cars from the simulation. *White* circles are traffic light junctions that use a simple round robin scheduling scheme to change the active traffic light. *Green* circles are traffic light junctions with an active epiNet controller. The red bar visible on the road adjacent to the epiNet controlled system indicates how long traffic has been waiting on that road (useful to visualise for controller design). Yellow cars have no assigned route, and move randomly, whereas green cars have a pre-set route. The title bar displays the simulation time the simulation has been run for, the state of the simulation (paused, accurate time or fast) and instructions to change the simulation state.

.

its own set of traffic lights, and cars will only enter the system when their light is green. Systems can be controlled by a traffic controller, else they assume a round-robin configuration.

Objects can be modified by changing the three loadable files which define aspects of the simulation. A `.txt` file format was chosen to remain in line with the objective of being intuitive and to stay consistent with section 3.4 (where it was seen how epiNet implements this file type to store network data). The intent of the three files used in the simulation are as follows:

- *Design* - declares new locations and systems, and what their two dimensional coordinates are. Specifies which are connected together via roads.
- *Test* - declares if, and when, a location will spawn a car. Determines whether these cars will have a route, and what that route is. Test variables can be randomised over a desired scale.
- *Map* - used to map a system's controller inputs and outputs to epiNet.

Once the simulation has been declared and loaded, it can then be *run*. It uses the standard SFML functions and *game* loop (from [29]) with regard to this; each object is updated with all interactions handled and then each is drawn to the GUI window. The simulation treats each frame of animation as a unit of time. By controlling when frames are displayed, the simulation's speed can be controlled. A user uses the up and down arrow keys to move between fixed simulation speeds (paused, accurate time and as fast as possible). The time information is updated in the title of the window, with state information and controls. The simulation can be run multiple times, with or without visuals, and multiple windows can be open simultaneously. The behaviour observed is saved as a data metric (again in a `.txt` file) known as the measure of effectiveness (MOE). This simulation currently considers a singular objective: the total duration time of every car to complete its route. Simulations are usually repeated with new random inputs, or *seeds*, to validate their findings.

5.3 Code Structure

The general structure of the C++ program is given in figure 5.2. The simulation class, using the three load files (discussed in section 5.2), creates location, system, road and car objects as required. These objects are stored in a vector (a class of sequence containers in linear arrangement) defined in C++'s standard library (found [25]). They also inherit an appropriate SFML class to acquire methods required to display each graphically. The simulation class then handles the drawing and updating of objects it has created. The location, system, road and car objects are also made aware of one another through forward declaration such that they can access functions from one another. Location and system inherit the virtual *Node* class such that Node containers, which can hold both separate classes, can be used. This is useful when forming connections between node-based classes. *VarClock* is a timer which uses frames as its tick increment. This is useful for Node-based classes which require specific timings; stating a delay for five seconds using *VarClock* is more readable than explicitly stating how many frames must pass for instance. Lights is a class which draws the current state of traffic lights on the road and is determined by looking at the state of the node the road end is connected to. Route is a class which holds a sequence of desired nodes (as pointers) that a car wishes to visit. It also holds the duration of the route. The Measure of Effectives (MOE) class is responsible for determining the fitness of the simulation. Main contains the entry point for the program and builds and runs simulations.

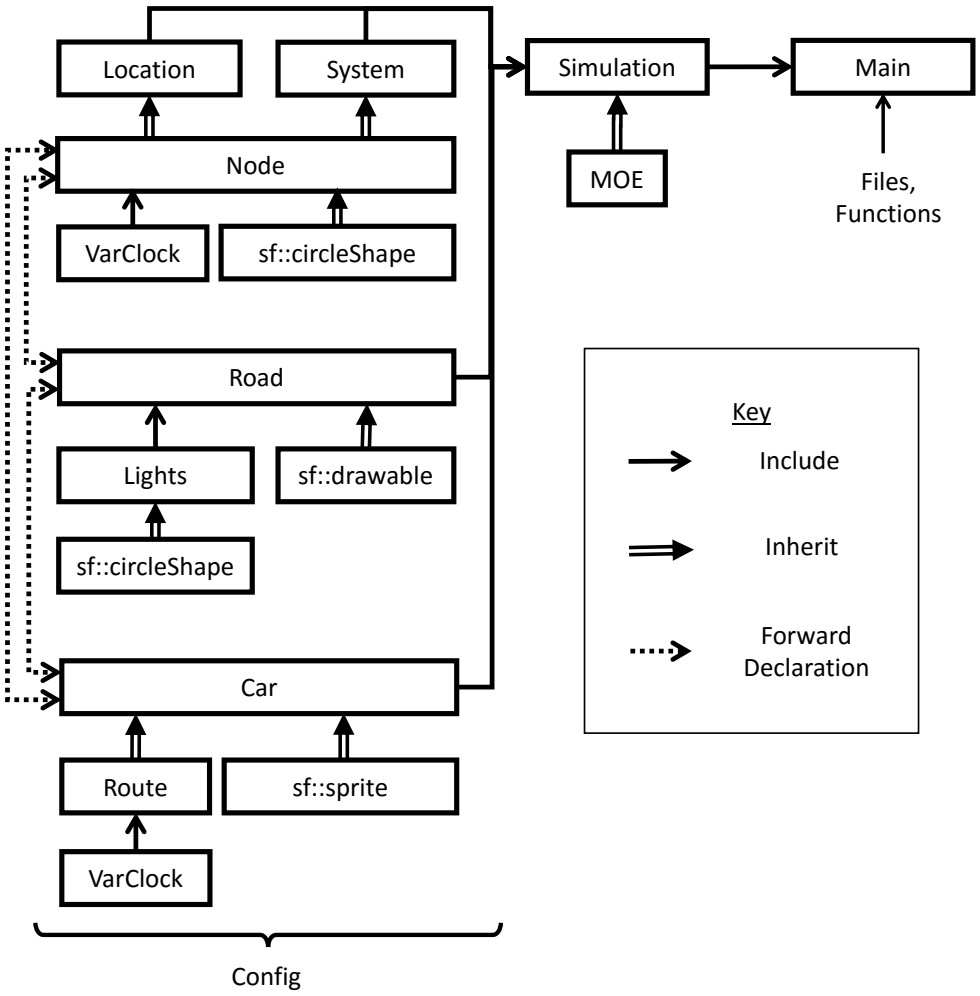

Figure 5.2: Diagram illustrating how classes interact in the C++ code for the agent-based simulation. Square boxes represent classes. Classes preceded by sf:: specify that the class is part of the SFML library. The key can be used to determine whether a class is included, inherited or forward declared by the class it points to. The unbounded labels are used to illustrate how non-class files interact with the structure; *Files* and *Config* are header files specifying important system macros and *Functions* is a set of high-level functions intended for use by novices.

5.3.1 Simulation Loop

Once the simulation has been declared and loaded, the run function (psuedocode given by algorithm 2) can be executed. Note that this pseudocode seeks not to detail every intricacy of the function. The function is a classical game loop, based upon the synchronised coupled model [30], which runs until the simulation finishes or is exited manually by the User. The simulation treats each cycle of the game loop, or frame, as a unit of time. Objects move with respect to this frame. For instance, between frames a car will travel a distance depending on its speed. When the GUI is not required many aspects of the run function are removed to improve efficiency. This includes removal of the render window, all drawing functions and removing User input. The Simulation is also forced to run as fast as possible by removing its

Algorithm 2 Simulation class's run function

```
1: Create render window from SFML library.
2:
3: while the simulation has not finished do
4:     if not ready to draw next frame then
5:         sleep thread until ready to draw frame.
6:     update total simulation time.
7:     if User inputs a command then
8:         react to that command.
9:     Update each simulation object.
10:    Clear render window.
11:    Draw each simulation object to the render window.
12:    Display render window.
13:    Determine if simulation has finished.
```

ability to sleep.

5.3.2 Simulation Update Routine

On each iteration of the simulation loop, all simulation objects (cars, roads, locations and systems) need updating. This begins by updating the states of each node and allowing traffic lights to react. Next car positions are updated. This is a complex process which will now be briefly considered. To understand the methodology first consider figure 5.3. From this it can be seen how each node stores the pointer to a car. This, in the first instance, will be of the form of a location spawning a car by grabbing the pointer to the next car in its associated spawn list. Now present in the simulation the car moves toward the location's connection with a road that leads to another node the car wishes to visit (either given by its internal route else set randomly by the location). The car pointer is then removed from the location and added to the correct road vector as it transitions onto that road. Multiple cars can be added to the vector and their order is always preserved. The advantage of such a data structure is that only the head car in the vector needs to check the state of the traffic lights in front to determine its subsequent behaviour. Cars following need only to check the state of the car directly in front of them (relatively much less complex), and queueing traffic needs only the head of the queue to be checked. The combination of these points save much computational time, especially when the simulation experiences dense traffic. Cars will remain in the road vector, slowly moving towards the head of their queue as other cars are removed. Once there, and assuming their traffic lights are green, the car will be removed from the vector and added to the node they have driven towards. This entire process is repeated until the car reaches another location and its pointer is removed from the simulation.

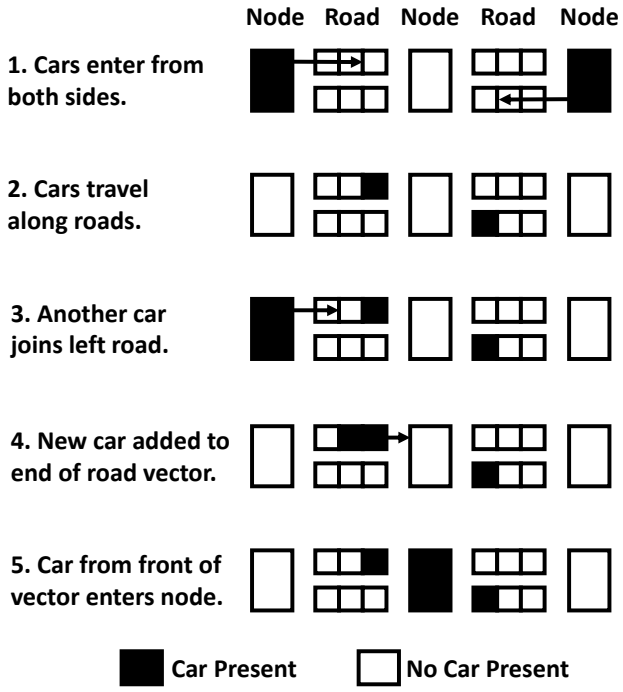

Figure 5.3: Diagram illustrating how (pointers to) cars are stored in data structures between nodes and roads. The diagram first shows two cars entering from the outer nodes which hold a singular car at maximum. These cars are then passed to a road vector on the correct side of the road they are joining. It can be seen as another enters, it is added to the back of this vector. Cars are removed from the front of the vector first.

Though this process it intuitive and relatively quick, it means cars cannot drive themselves; their movement is handled by the structure (currently pointing to the car) as it calls simple movement commands within the car class. This ultimately makes creating more natural human behaviour, which could lead to the emergence of interesting traffic characteristics and thus new research avenues, more difficult. The approach described was ultimately chosen however to remain more intuitive and reduce complexity. The reasoning for this preference is unchanged from the introduction of this chapter.

5.3.3 Car Traversal

First consider figure 5.4, which shows two cars driving in opposite directions on a road. Cars have Cartesian coordinates and a direction they face (Note that SFML measures angles clockwise). The coordinates are centred centre-right of the car such that driving on roads, and turning, are easier to compute mathematically (cars drive on one line between road ends and turning requires only rotation). By updating these parameters the car *drives* around the system. The global coordinates are centred at the top left of the render window with x and y axis taking their standard directions.

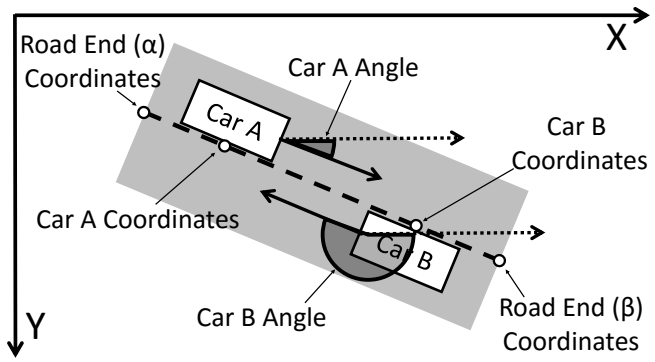

Figure 5.4: Diagram depicting two cars (*Car A* and *Car B*) travelling on opposite sides of a road (Road between α and β) which exists in a two dimensional space (the render window). Both cars are shown with opposite heading angles (angles are drawn from the x-axis). The diagram shows how by centring the car's coordinates on the center-right side of the car, that both cars can drive down the same path while appearing not to collide.

Each pixel is a distance in either the x or y dimension, hence moving the car one in the x direction moves the car one pixel to the right. Cars however are required to move forward with respect to their current heading. To achieve this equation 5.1 is required to determine the path's unit vector ($\hat{P}$) where Δx and Δy are the differences in x and y positions between the car and its desired location respectively.

$$\hat{P} = \left(\frac{(\Delta x)}{\sqrt{(\Delta x)^2 + (\Delta y)^2}}, \frac{(\Delta y)}{\sqrt{(\Delta x)^2 + (\Delta y)^2}} \right) \tag{5.1}$$

The unit vector is calculated once and the result stored. This is done because, as it will not change on a straight road, the vector can be re-used saving much computational complexity. $\hat{P}$ is then multiplied by the speed of the car to work out the vector the car needs to move. This process is known as the `drive()` function and resides within the Car class. Speed has a value much less than one such that (considering tens of frames are drawn every second) car animations remain smooth (cars don't move too far between frames that they appear to jump).

Systems are more complex; they drive a car into their centre, turn the car and drive it onto its desired road. These states take the form of an enumerated type (with states called: *entering, left/right and leaving*). When a car is passed to a system, the system uses the aforementioned `drive()` function to move it to its centre. Once there the angle which the car needs to turn, and whether the turn is to the left or right, needs to be determined. To understand how this is achieved first consider figure 5.5. It shows how the difference between the car angle and the desired car angle is used to determine the angle to turn. If this angle is less than 180° the car turns right, else it turns left. Note that negative rotations and over rotations are

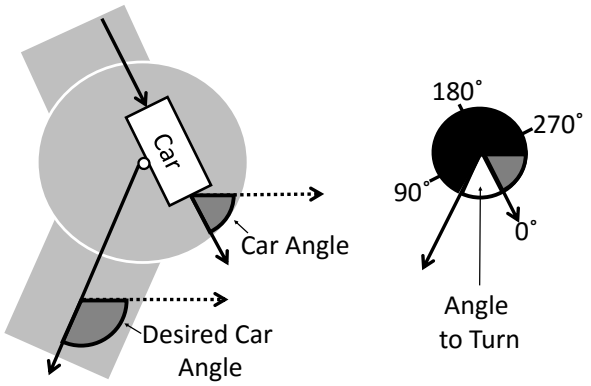

Figure 5.5: Diagram depicting a car that is just about to begin its turn within a system (shown on the *left*). The car's angle is shown as is the desired car angle. The difference between the two gives the angle that needs to be turned. This is shown on the compass (to the *right*). Note that if the angle to turn is between $0° - 180°$ the car will turn right, else it will turn left.

protected against by adding 360° (effectively doesn't change the angle) and using the C++ function fmod[1] to take the floating-point remainder when dividing it by 360. This leaves the same effective angle, but guarantees that it is in the range of 0° - 360°. That angle is not instantly turned, instead the state is changed to left or right depending on which direction is required. That angle is then turned in a pre-defined iterative step until the turn is complete (over multiple frames thus animating a turn). Once facing the new road, the state is changed to leaving and the drive function is re-used to drive the car onto the road.

5.3.4 Object Collision

In section 5.3.2 object collision was detailed. This section looks to build upon this introduction to explore the mechanics implemented more thoroughly. The only traffic environment, in the simulation, where cars need to observe their surroundings is when driving on a road. The other environments, systems and locations, can only have one car present within them and this car has already passed through the traffic lights (present on the adjoining road). There is therefore nothing to *collide* with. On the road, as introduced and reasoned in section 5.3.2, there exists two possible collisions that must be checked:

1. The head car on a road needs to check the state of the traffic lights in front.
2. Subsequent cars need to check the state of the car directly in front of them.

Considering the first point. The head car on a road drives towards a road end. If, while travelling at its current speed (which may be increasing toward the speed limit of the road),

[1] `fmod` source: `msdn.microsoft.com/en-us/library/20dckbeh.aspx` [Accessed: 02/05/2016]

it is calculated that the car wouldn't reach the road end, then the end traffic lights are not checked. If it will reach the road end the lights are checked and the car either stops at the road end, if the lights are red, or continues on (if green). This process is illustrated in figure 5.6. The prediction, of whether the car will reach the lights, is calculated by taking the magnitude of the vector between the car and the destination, and comparing it to the speed of the car. If the speed of the car is less than this magnitude then the car will not reach the lights. If the head car has stopped at the lights, it will check the light's state every frame until they change (and it can enter the adjoining system or location).

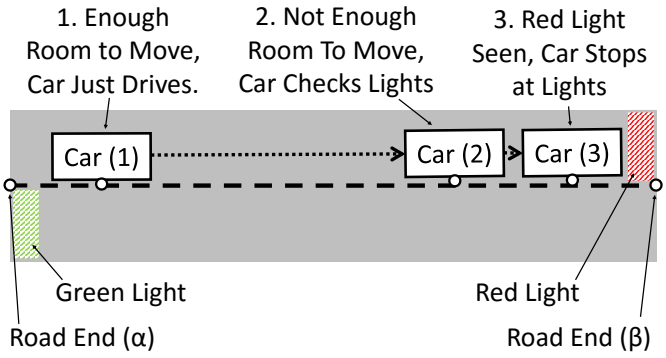

Figure 5.6: Diagram illustrating when a single car on a road (or the head car) checks the traffic lights at the road end. It is broken into three stages. In the first stage the car (*car(1)*) determines there is enough room to move before colliding with the traffic lights so it doesn't check the lights, it just moves. On the next frame the car is now at position *car(2)*. Again it checks if there is enough room to move before the lights. It calculates that there is not and so it must check the state of the lights. It sees the lights are red, so the car moves to the lights and stops (at position *car(3)*). On each subsequent frame the car will check the state of the lights and will only drive into the adjoining node when they turn green.

Consider now the second collision case. A trailing car is updated after the car in front. The car in front also never slows down such that it is either travelling at least as fast as its tail or it has stopped. Thus cars that are not heads need only check if the car in front has stopped. If not, they can accelerate without fear of collision or hitting the road end. If the car in front has stopped, then the trailing car uses the same technique as was described for approaching traffic lights with one difference. That is the car needs to stop a car length (average of both car lengths to accommodate cars of differing lengths) away from the car in front such that their bodies do not collide (plus a small constant so there is more realistic spacing). This is achieved, when the trailing car does the comparison, by: taking the car's direction (a unit vector); multiplying it by the desired car separation; and subtracting it away from the leading car's coordinates. The speed of the car in front is also considered in this calculation. This changes how queuing traffic moves along roads. Faster roads will lead to queues moving off at higher speeds, and thus the separation term will become bigger, leading to cars leaving wider

separations (mimicking stopping distances).

Cars cannot decelerate (brake) in this simulation which is incredibly unrealistic. They accelerate to a maximum speed and when something can be collided with they stop instantly. However, this approach massively reduces the number of collision comparisons conducted on each frame. This allows simulations to run much more quickly. Therefore, in line with the justification presented in the introduction of this chapter, efficiency was consciously chosen over realism.

5.3.5 Intergration of EpiNet Controller

The simulation allows for systems to be controlled by epiNet (introduces in chapter 3). The User is able to do this by calling a simulation constructor (through polymorphism) that can take a vector of epiNet network pointers. The inputs and outputs (IOs) from each of these networks are then stored in their own vector. When the simulation is being built from the three set-up files, discussed in section 5.2, these IOs become mapped to individual systems. This mapping is very simple; the map file only states how many IOs from each network are mapped to each controller (for instance, input 1 and input 2 of epiNetwork 1 map onto system number 1 and so on). Protection exists in the code to prevent incorrect mapping. It also allows for inputs and outputs of a system or controller to be unconnected. The inputs of a controller (the outputs from epiNet) are commands the controller must perform. The outputs of a controller (the inputs to epiNet) are the current state of a traffic system. The current best configuration of IOs is explored later through testing (in chapter 7). The update routine is now also required to update a traffic controller. Note that a controller is not updated on every frame, but rather a large pre-defined multiple. This is to disallow the controller an unrealistic reduced computational latency that would unfairly match the state machine approach of a round-robin. This latency is the product of the processes a complex controller must perform: wait for all road sensors to be read; process the information received internally (this may even be dependent on other epiNet systems); and communicate to connected traffic systems (there may be more than one, and they may not be local).

5.4 Problems Encountered

"We cannot perfectly predict a software development project" [31]. The source later justifies this noting: inevitable obstacles; how ideas change; and that software is inherently intangible.

The development of the agent-based simulation certainly supports this position, and this section seeks to highlight this by introducing the key compromises.

5.4.1 Stack Overflow

Local variables are stored on a computer's internal stack. The simulation's constructor therefore is kept relatively simple as to generate all objects within it could exceed the capacity of the stack (called stack overflow). This in reality is unlikely to happen and thus not usually considered, but considering the simulation is unbounded and must hold every program object it is not impossible. By declaring vector objects outside the constructor, such that they are allocated to the heap, this situation, however rare, is avoided. This comes at the compromise of the simulation's run function requiring extra time to build objects before it begins which provides noticeable initial operational delay.

5.4.2 Communication Between Simulation Objects

A problem arose when moving objects from being stored in arrays to vectors to allow for greater flexibility in the program. It was found that roads no longer formed connections between any two nodes. Ultimately this was due to the use of referring to road objects (using the keyword *this*) in the constructor of that object. That is, the road constructor passes a pointer of itself to the two nodes it connects to in its constructor. This is bad programming practise and ultimately behaved in an array implementation through luck. The code was changed to remove the use of *this* from the constructor, thus solving the problem, but this was a quick fix. Objects still pass information between each other in a convoluted manner through forward declaration. With hindsight, a better approach would have been to use a higher-level class that manages the connection of, and communication between, simulation objects. This would lead to more intuitive and readable code. Ultimately this has not been conducted due to the time constraints of the project; research avenues are more profitable than the optimisation of the simulation tool.

5.5 Summary

This chapter presented the agent-based simulation developed in this project. This began with the following justifications: graphical simulations are easy to demonstrate and debug; using the SFML library allows for easier and safer development; and reducing the complexity

of simulations allows them to be computed much more quickly. The functionality of the simulation was then outlined. Cars are spawned at locations, travel along roads and through systems, until they reach another location where they are removed. Performance is measured by how long cars take to pass through the simulation. Three files were seen to control: simulation design, car spawning and routes and controller mapping. The structure of the code, and problems encountered, were then presented with the main point being how epiNet could be used to control the traffic lights at traffic systems.

The next chapter introduces how evolutionary algorithms can be used to improve an individual over time. Once this has been presented, subsequent chapters are then able to use this technique to evolve an epiNet-based controller upon the simulation described in this chapter.

Chapter 6

Evolutionary Algorithms

Contents

This chapter introduces Evolutionary Algorithms (EAs) based upon the work of Yu & Gen in [32]. This is achieved by first presenting background biological knowledge of Darwinian evolution. From this the general process of an EA is stated with key parts subsequently expanded upon. Definitions of two sub-categories of EAs, which are required later in the report, are then given. Finally a concept known as over-fitting is explained.

6.1 Darwinian Evolution

The section introduces the process of evolution focusing mainly on how an individuals set of genes change over time to adapt its phenotype to the environment. Darwin's, "The Origin of Species" [33], famously observed that an individual organism is more likely to survive if it develops traits which make it more adapted to its environment. Dawkins later, in [34], furthered this argument by suggesting that this concept is applicable to individual genes. That is genes which produce improved phenotypes result in an individual being more likely

to survive and thus more likely to pass on its genetic information to future children over its competitors. This process is known as *natural selection*. An individual's genotype tends only to be altered at its point of creation where it usually takes half its genetic information randomly from both parents [16]. At this point some random mutations to the genotype may also occur allowing natural selection to develop the emergence of complex behaviours [34].

6.1.1 Cooperative Evolution

The section looks to introduce Darwin's work on *co-evolution*, first mentioned in [33] and subsequently in [35]. Co-evolution is the act of two, or more, separate evolving genotypes influencing each others development. The canonical example being that of predator and prey where, as the predator evolves to prefer individuals better at hunting, the prey evolves to prefer individuals that can best combat this new threat. This cycle can continue indefinitely. Co-evolution has been found to be both a rapid process that can radically change a genotype within only a few generations, and a dominant one thought to affect the vast majority of known life [36].

6.1.2 Hierarchy

This section seeks to introduce the biological principle known as *hierarchy* based on the work of Allen & Thomas [37] and Gould [38]. Hierarchy is defined as a system of communication, where entities are defined by the extent to which they constrain or filter incoming information. In terms of genotypes, this can be represented by individuals sharing access to, or portion of, the same genotype. In this way individuals, which may have different needs, must evolve in a way which is cooperatively good for both individuals at each generation. Note that individuals can maintain significantly differing phenotypes even when sharing similar genetic information. This structure is currently only found within single organic entities and not between different organisms. One notable example can be found in humans; although there are around 210 cell types in the human body each share the same genetic information (DNA) [39].

6.2 General Process

This section looks to introduce Evolutionary Algorithms (EAs) and describe their general process. An EA is essentially optimisers which look to, by trial and error, find a set of inputs which produce the most ideal behaviour within a system or function. Their name stems from the parallelisms that can be drawn to Darwinian evolution, discussed in section 6.1, in the way in which they approach this optimum:

- *Population-based* - An EA maintains a set of solutions, or a *population*, to the problem they seek to optimise.
- *Fitness-oriented* - A fitness function deduces how well an individual solves the problem (their fitness).
- *Variation-driven* - Populations undergo operations (discussed later in section 6.4) that mimic genetic changes. These are vital for allowing the algorithm to find new solutions (known as moving through the *search space*).

Leading from these principles the general EA process (illustrated in figure 6.1) can be formed. First an initial population must be formed. Then begins a cycle where: each individual's fitness is evaluated; the best parents are chosen; children are created from said parents (with mutations). This process can be terminated at any stage as desired with time constraints (to avoid endless program execution) or a required fitness value usually being the termination condition.

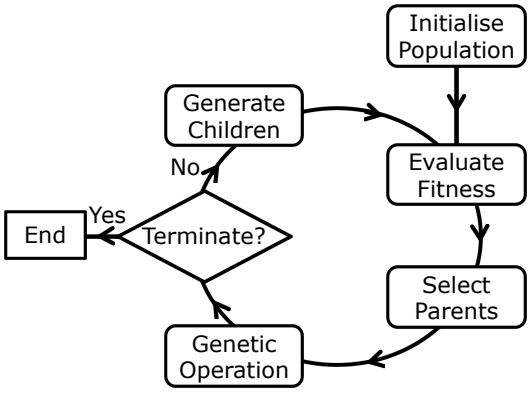

Figure 6.1: Flow chart showing the operation of a general evolutionary algorithm (loosely based on a diagram from [20]).

6.3 Natural Selection

As discussed in section 6.2, new generations are formed from the fittest individuals from the previous generation. Three factors determine the new population:

- The number of parents (denoted by λ)
- The number of children (denoted by μ)
- Do parents pass to the new generation (denoted by ',' for no and '+' for yes)

In general, the fittest individuals are required to become parents to move the next generation towards an optimum. However, if multiple optimums exist close together picking just the fittest, known as *tournament selection*, may restrict the movement of the algorithm. A better approach may be to pick a more diverse set of parents (as seen in figure 6.2). Determining the optimum number of children results in a compromise between whether the EA requires a larger local search space for each generation to prevent becoming stuck at a local optimum or whether the EA is required to converge quickly. Why this compromise exists is best understood with reference to figure 6.3. In general parents are usually kept within an EA. This is so that the fittest solution always survives (known as elitism). Some cases that prefer μ, λ evolutionary algorithms, known as *generational* EAs, do exist however.

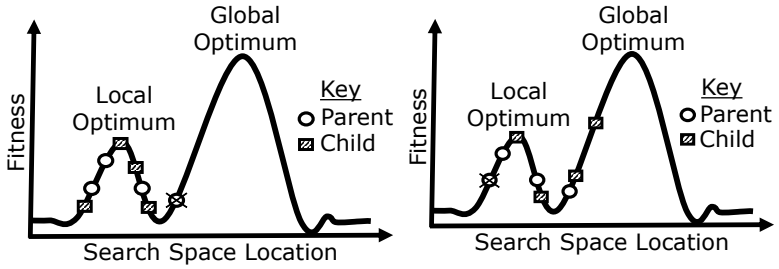

Figure 6.2: Diagram showing how picking just the fittest parents can increase the chance of algorithms becoming stuck at local optimums. Specifically, the first graph (*left*) shows how picking the three fittest parents leaves the children stuck next to the local optimum whereas the the second graph (*right*) shows how picking a more distant parent has allowed children to move away from the local optimum and towards the global one.

6.4 Mutation and Crossover Strategies

This section introduces crossover and mutation strategies with regard to EAs. Crossover is the process of combining the genotypes of two parents to form a new child. The main

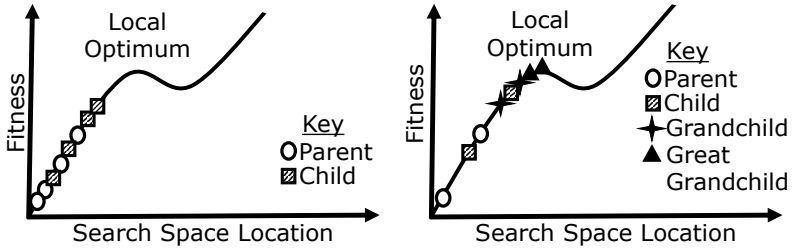

Figure 6.3: Diagram showing how reducing μ and increasing the number of generations can help populations find optimums more quickly, but reduce their ability to escape local ones. More specifically, the first graph (*left*) shows how a large generation size cannot even find the local optimum, whereas the second (*right*) shows how many generations found the local optimum quickly but then became stuck.

strategies explored in this section are that of *N-point crossover* and *uniform crossover*, though many more strategies exist. N-point crossover is where the child takes genetic information from one parent until a pre-defined crossover point where they swap to the other parent to continue copying genetic information. The N refers to how many crossover points exist. Generally crossover points are placed at equal distances apart through the genotype, though this need not be the case. If the crossover happens at different places in both genotypes the process is referred to as *cut and splice* and will likely result in the child's genotype being of a differing length to the parent. Uniform crossover takes a predefined mixing ratio, and randomly splits the parent's genotype within this ratio to form the child. Crossover can be non-essential or counter-productive in some applications. Mutation, the random altering of an individual's genotype, is crucial in allowing subsequent generations the ability to move further towards a local optimum value within a search space. Many mutation strategies exist within EA, but in general bits within the genotype, whether they be random or selected in some way, gain some probability, or *mutation rate*, of being flipped. Mutation rates can be fixed or dynamic. The main concepts introduced in this section are shown in figure 6.4.

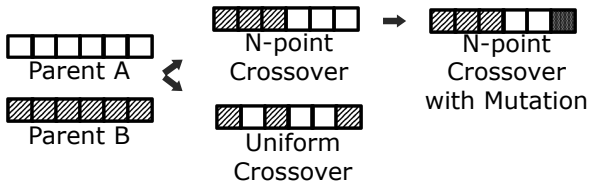

Figure 6.4: An illustration demonstrating the different ways in which children can be formed from their parents. Specifically, N-point crossover (where $N = 1$), N-point crossover (again, where $N = 1$) where the final portion of the child's genotype has mutated, and uniform crossover (with mixing ratio of 0.5).

6.5 Evolutionary Strategies and Genetic Algorithms

Within the broad definition of Evolutionary Algorithms there exists sub classes as described in [40]. The project presented requires the introduction of two: Evolutionary Strategies (ESs) and Genetic Algorithms (GAs). An ES tends to hold a small population (four or five being common) and tends to pick a single parent in tournament selection. Crossover is therefore rare in an ES as their parent tends to be simply cloned and mutated to form children. A GA holds a much larger population (typically thirty as a minimum). Natural selection tends to be modelled more accurately in a GA than seen in section 6.3. Firstly, a GA tends to maintain more individuals between rounds (typically fifty percent). They also tend to have a selection operator to pick a parent. As an example, they might pick a parent from a generation by picking the fittest from a set of randomly chosen individuals. This means that the fittest individuals in a generation, while more likely to become parents, are not guaranteed. Additionally crossover is more extensively used to generate new populations. In general an ES is quicker at finding an optimal solution than the more complex GA. However, due to a genetic algorithm's better perspective on population dynamics compared to an evolutionary strategy, a GA tends to be more able to navigate complex search spaces.

6.6 Overfitting

This section explores overfitting and how it can adversely affect evolutionary algorithms. At each iteration of the evolutionary process the genotype of an individual is refined such that its phenotype is better adapted to its environment. As the process continues the genotype becomes more specialised to the set of data from which it is learning. Excessive specialisation of the genotype can leave it unable to handle new information. To understand this consider an arbitrary example where a controller is being evolved, by listening to a specific musical track, to record audio. Initially the population may begin to filter the upper and lower ends (as this gives the biggest fitness augmentation), but soon additional filters between frequencies will be preferred as the genotype learns how to fit filters exactly to the track (to achieve those final small fitness improvements). If the controller is presented with new music some regions of the track now may be missing. This is a classic example of over-fitting. This may not of happened if the controller was used at an earlier stage before the final filters were added (the controller is not over-trained). This could have also been avoided if the track randomly changed between

tests (while retaining the same generic features) such that multiple generations with identical training data, where filters could be over-fit, are no longer present.

6.7 Summary

This chapter introduced EAs. This was achieved by first considering Darwinian evolution before considering two less familiar cases: co-evolution and hierarchy. The general process of an EA was then outlined. It was seen to be a cycle whereby a population: is expanded through random mutation or crossover; tested through fitness functions; and the weakest removed. Two main variations of this methodology were then presented: ES and GA. An ES was seen to hold a smaller population and thus converge upon a solution quicker. A GA was seen to hold a larger population, with more complex sexual selection operators, and thus able to navigate more complex search spaces without becoming stuck at local optimums. The importance of not over-training individuals, or ensuring they are tested on randomised fitness evaluation tests, was also considered.

The next chapter formalises an EA methodology as it evolves an epiNet-based traffic controller by implementing it on an agent-based traffic simulation. Later in the report the concepts of co-evolution and hierarchy will be revisited as multiple epiNet-based traffic controllers are evolved in a closed environment.

Chapter 7

Evolving a Single EpiNet Traffic Controller

Contents

In chapter 5, agent-based modelling was developed and it was seen how epiNet was integrated as a traffic controller. Chapter 6 then introduced evolutionary algorithms (EAs) and how they can be used to iteratively optimise a system. This chapter expands on these to detail investigations undertaken in the development of a methodology to evolve a single epiNet-based traffic controller.

7.1 General Process for Evolving EpiNet Controller

This section presents the general way in which an EA was implemented to evolve and an epiNet traffic controller using the agent-based simulation. The general process is given as psuedocode in algorithm 3.

Algorithm 3 General flow of Evolutionary Algorithm to Evolve EpiNet Traffic Controller

```
1: Create or load initial epiNet network into population.
2: Fill population with random epiNet networks.
3:
4: for The number of generations do
5:     for The number of tests per generation do
6:         Create standardised test from random template.
7:         Test controller using test on desired simulation.
8:
9:     Calculate average test duration for each individual in the population.
10:    Select Parents.
11:    Create New population.
12:
13:    if Controller needs saving mid-operation for this generation then
14:        Find fittest individual in the population.
15:        Save epiNet network to file.
16:
17:    if Termination condition is reached then
18:        Break from generation for loop.
19:
   return Fittest individual.
```

The main point to note about this implementation is that, as seen in section 6.6, multiple fitness tests are conducted for each individual in the population to prevent over-fitting. To achieve this the desired training data uses randomised values. Then, for each round of testing, these values are fixed using a random seed. This ensures that each round of testing is consistent between individuals, but testing between different rounds are randomly different.

7.2 Four Way Junction

Tests conducted in this chapter refer to a four way junction. This section therefore seeks to introduce this junction and how it behaves within the agent-based simulation. First consider the screen shot of the four way junction given in figure 7.1. In the centre of the junction exists a traffic system with 4 road connections. These connections lead to locations that can spawn and destroy traffic. The system is controlled by an epiNet traffic controller (hence the green colour). For the system to be efficient, such that the total duration of all cars to pass

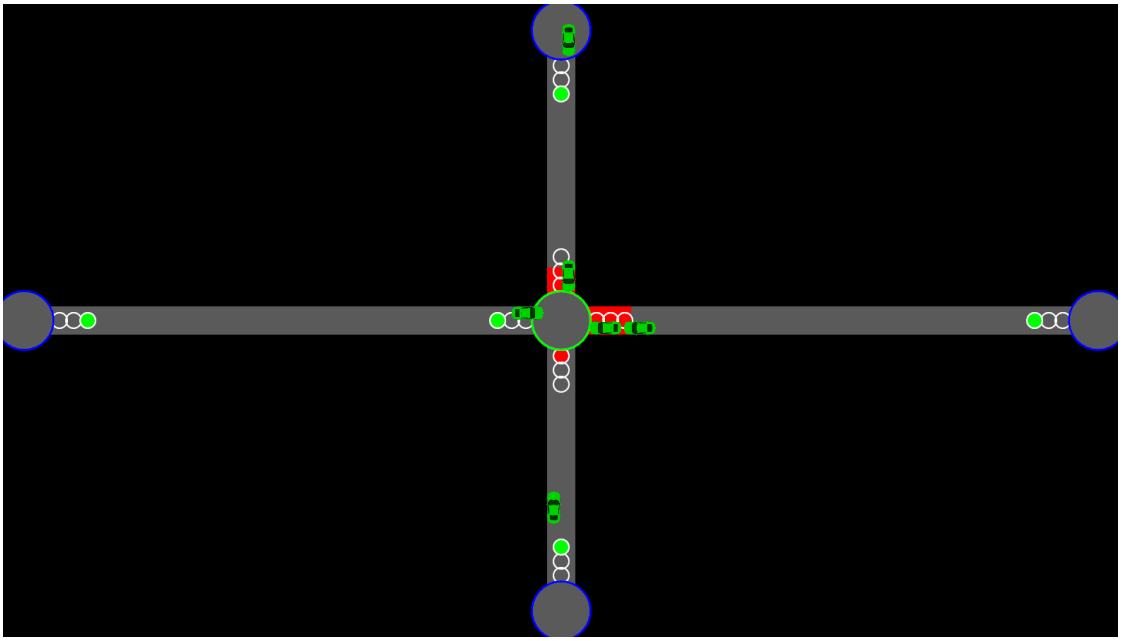

Figure 7.1: Screenshot from the agent-based simulation's GUI as it displays the four-way junction.

through the system is reduced, it must be able to efficiently swap between roads containing traffic. The system is incredibly simple and symmetrical, and so for even traffic distributions, a round-robin controller can work very effectively. Additionally, epiNet is a complex traffic controller that is not well suited to this situation as complexity adds latency issues as discussed in section 5.3.5. This is inconsequential for the tests conducted in this section however; the methodology is important at this stage not the final quality of the controller.

7.3 Evolutionary Algorithm Choice

This section compares two methods for evolving epiNet: An Evolutionary Strategy (ES) and a Genetic Algorithm (GA). The methodology of how both were technically implemented and tested is first given, before results are presented and discussed.

7.3.1 Methodology

In section 7.1 the general structure of the EA implementation was presented. Differences exist however within this methodology between ES and GA. Consider table 7.1 which outlines the main differences between those implemented. The initial epiNet network was standardised between testing of both EAs with a fixed number of 30 genes and 3 epigenetic molecules.

Table 7.1: Main properties of the ES and GA implemented

	Evolutionary Strategy	Genetic Algorithm
Population Size	4	30
Test per Gen.	7	7
Parent Selection	Fittest individual	Fittest 50%
Reproduction Methodology	1. Clone parent with mutations. 2. Repeat 1. till complete.	1. Take fittest from random three. 2. Repeat 1. 3. (70% chance) create two by uniform crossover of 1, 2. 4. Mutate both from 3. 5. Add 4. to population. 6. Repeat 1, 5 till complete.

The input and output ratios were set to 0.5 and 0.4 respectively. These values were found by experimentation; they are a good compromise between evolvability (more genes provide more complex mathematical possibilities) and speed of convergence (less genes result in a smaller search space). The epiNet controller was set to take 4 inputs mapped to whether each road had a car waiting. This value begins at 0.5 when a car is first present and slowly increments to 1 over a pre-defined period of 10 seconds (of simulation time). The outputs were mapped to control the traffic lights of each road entering the junction whereby the largest value is chosen and the corresponding road is activated. The input and output settings have been arbitrarily chosen at this stage. The mutation rate for both EAs was fixed at 5%. This value was chosen by suggestion of [32] as to allow both EAs good ability to evolve through a noisy search space, while not being so large as to transition the behaviour of both towards random search. Both Strategies were given 200 generations to evolve. This value allowed for both algorithms to converge on an answer (converged within 25 generations) such that the greater number of individuals checked by the GA could be negated as a factor if it was to outperform the ES. The test conducted, in terms of physical cars driving through the simulation, saw 3 cars spawned from each location with random 10 second intervals (the last car can be spawned at 40 seconds into the simulation at the latest therefore). This test was found through experimentation. It is a good compromise between being quick to run while randomly exposing the system to sparse, moderate and heavy traffic. This test is used to both train the EAs (fitness function being the total duration of cars to pass through the simulation) and then test each controller (50 times).

7.3.2 Results

The results from the test described in 7.3.1 are shown graphically in figure 7.2. They show

that the ES was unable to evolve a controller that could efficiently control traffic and that the GA was able to evolve a controller comparable to the round robin. On closer inspection

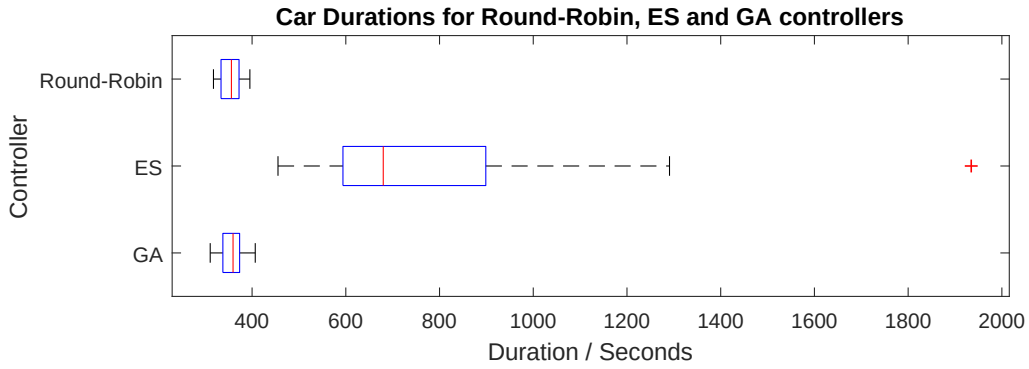

Figure 7.2: Box plot showing how long, in seconds (lower is better), it takes for cars to pass through the four way junction for different controllers (Round-robin and epiNet controller evolved using an ES and a GA). The graph shows the ES was unable to sufficiently evolve a controller whereas the GA evolved a controller comparable to the round-robin. Round Robin mean = 355, GA controller mean = 359 , ES controller mean = 680.

of the ES controller, it could be seen graphically in the simulation that it had evolved to ignore a single road of traffic at random. Evolution of controllers using the ES in two further tests found the same behaviour to occur. It is therefore likely that, at least for the four way junction, that the search space for traffic simulation has prominent local optimums as this would explain why the GA was able to converge on a better solution (reasons discussed in chapter 6). The salient conclusion arising from this testing is that a GA is required for evolving epiNet traffic controllers. This is unsurprising as it supports the findings of Turner [20].

7.4 Optimising Controller Inputs and Outputs

This section expands upon the methodology of section 7.3.1 and attempts to ascertain what inputs and outputs a traffic controller should have. Ultimately this was determined through a combination of experiment, intuition and the previous work of Turner et al. (in [20], [2], [1]). This section therefore presents two primary cases, the worst and best case, that conclusions can be drawn from.

7.4.1 Methodology

Changing how inputs and outputs are used within the agent-based simulation required explicit modification of functions within the system class (the class that controls how the junction

behaves and what it does with each input from a controller). The first case took the same controller design from section 7.3.1 and added an additional input, which road is currently green (n roads encoded on single input), and an additional output for how long the switched to light should remain green (output multiplied by constant of 10 seconds gave time to wait). In total therefore the first case has increased the complexity of the controller to 5 inputs and 5 outputs (or 5 in 5 out as it is more commonly referred to). The second case instead reduced the outputs by binary encoding them. Therefore it has 4 inputs and 2 outputs (or 4 in 2 out). A greater level of encoding than binary was considered, but rejected. This was because initial testing showed that epiNet could not handle such specificity on its outputs. This is likely due, as seen in chapter 3, that a sigmoid is used to process gene values. And therefore the operation is easier to split into boolean logic (both halfs of the sigmoid) than it is to portion it evenly between n cases. Testing of both conditions is identical to that conducted with the GA in section 7.3.1 except the total number of generations was reduced to 30 (only 25 were required previously and reducing the number of generations significantly reduces the computational time required to execute). And, as fewer generations are required, simulation test durations have been increased to expose systems to longer traffic intervals. Each test was effectively doubled from section 7.3.1 (twice as many cars over a doubled time period). It is expected that longer testing durations will improve the final controller performance.

7.4.2 Results

The results from the test described in 7.4.1 are shown graphically in figure 7.2 along with the results for a round robin and the controller developed by GA (4 in 4 out) in section 7.3.1 as control tests. Note the 4 in 4 out was evolved again using the new longer tests to ensure this factor does not skew any findings. The tests show that increasing the complexity of the inputs and outputs negatively effects the performance of the controller, whereas reducing the outputs and ensuring inputs are not encoded increases performance. Figure 7.4 shows the simulation durations for the 5 in 5 out controller as it evolved over time. Bars are only present when a new individual in a population is found to be the fittest. It appears to show the controller diverging away from its initial value over subsequent generations. This at first may seem unexpected, but is likely due to the stochastic nature of testing. It is credible to suggest that over more generations that the fitness may balance, or even begin to converge upon a solution. However, even if it could converge on a solution, this would consistently take too long to be a viable evolutionary process.

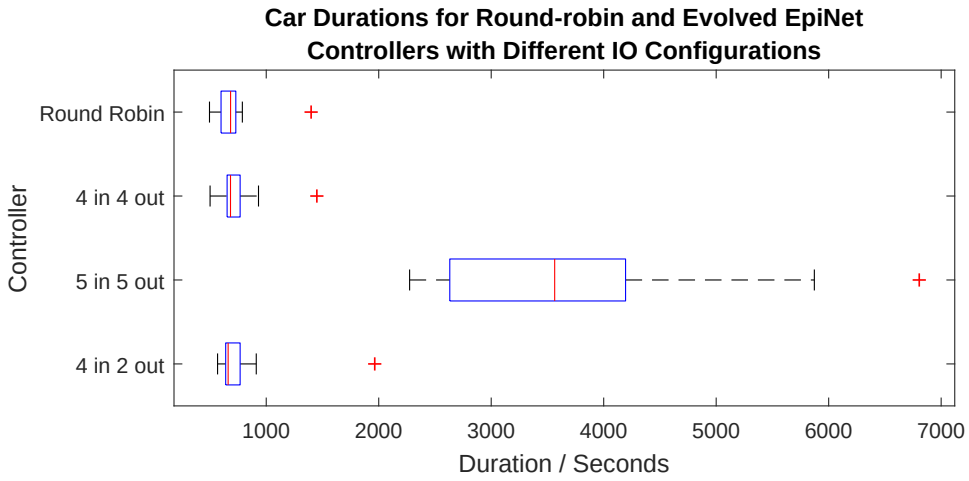

Figure 7.3: Box plot showing how long, in seconds (lower is better), it takes for cars to pass through the four way junction for different controllers (Round-robin and epiNet controller evolved using different configurations of IOs). The graph shows that reducing the number and complexity of IOs improves performance. Round Robin mean = 683, 4 in 4 out mean = 683, 4 in 2 out controller mean = 661 , 5 in 5 out controller mean = 3564.

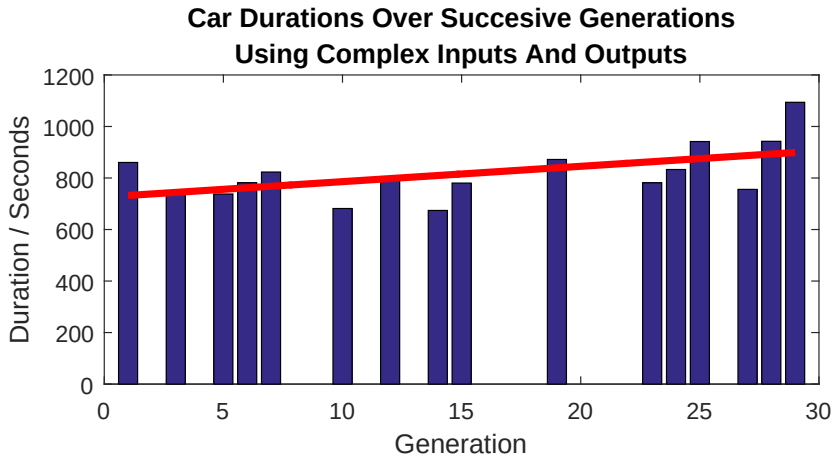

Figure 7.4: Bar chart showing durations (lower is better) for a controller using complex IOs (5 inputs and 5 outputs). Each bar represents the total simulation time for a random four way junction test when a new fittest individual is found. The graph shows that over time the fittest individual appears to get worse, though this is likely due to stochastic nature of the tests and the small generation size.

The results from these tests support the findings of Turner [20], whereby the difficulty of using complex IOs is noted. The results have also lead to a strategy for the configuration of IOs for the epiNet controller to be decided upon. The inputs should be mapped to whether a car contains a road, and how long that car has been waiting. The outputs of the controller should be binary encoded and mapped to individual roads.

7.5 Comparison of Methods to Average Fitness Testing

In section 7.3.1 the general structure of the EA implementation was presented. It was seen that, in order to accurately determine the fitness of each individual epiNet network, that multiple tests were required and the average taken. In general, the median is used to average tests within EAs as it neglects fringe cases which typically distort the average [32]. This section seeks to test the validity of this claim by comparing the use of median, used thus far, with that of the mean in averaging fitness tests with a GA.

7.5.1 Methodology

There is a requirement for two versions of the GA presented in section 7.3.1; one requires the median to average fitness tests (unchanged) the other the mean. The number of generations, as in section 7.4.1, have been reduced to 30 as both should have converged to a solution within this time frame. As it is important to see how both cases evolve over time, the fittest individual's average test result in the population (for each case) is tracked.

7.5.2 Results

The results of the tests (described in section 7.5.1) to determine which averaging method (median and mean) is superior are shown graphically in figure 7.5. There a three main features that can be identified in the graph. The mean: converges quicker than the median; has a less noisy evolution pattern; and converges to a smaller final value. To understand why this has occurred first considered that simulation tests are, considering there symmetrical and consistent design, difficult to randomly complete well. But there is plenty of scope for a controller to randomly perform poorly (it could ignore a road, be slow to react in some conditions and so on). This leads to fringe cases being disproportionally weighted towards bad cases which the median will ignore. The mean however punishes controllers that contain these poor cases. It is credible to suggest therefore that, as the mean preferences more robust individuals, that the GA holds a better, more robust set of genetic information as well. And that ultimately these traits are therefore carried across to the evolution process itself. The mean converges to a smaller value because the median method will never consider its poor fringe cases and hence will never attempt to improve them. There is a point of contention with these results that should be raised; the entire premise of these findings may be flawed. This is because the fitness evaluation (how the fitness of the controller is determined post

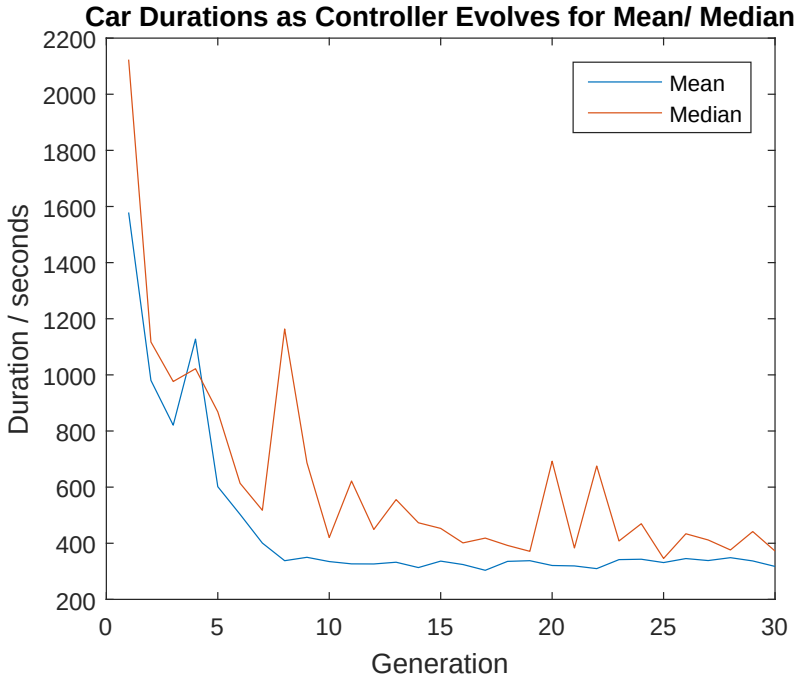

Figure 7.5: Graph showing how changing whether averaging, within the GA's fitness function, uses the mean or median effects evolution performance. The graph show that the mean (*blue line*) outperforms the median case (*red line*). The mean: converges quicker than the median; has a less noisy evolution pattern; and converges to a smaller final value.

evolution) determines the total duration that cars take to pass through a simulation by averaging multiple tests using the mean. Therefore a GA's fitness function that uses the mean is unfairly prepared for the fitness evaluation. To reconcile this further tests could be conducted to investigate if the same relationship holds true if the fitness evaluation is switched to using the median. However, though this has been considered by this report it has not been conducted. The reasoning for this being it remains true that fringe cases are disproportionally weighted poorly and as such the mean better punishes bad controller design. Hence, it is the view of this report that is should therefore be used in the fitness measure. This therefore leads to the conclusion that the mean should be used to average test cases in the GA.

7.6 Unbalanced Traffic

This section investigates how the epiNet controller developed in section 7.5.1 (using the mean) handles new, unseen traffic scenarios. This test is important; one of the main motivations for using bio-inspired controllers is their propensity to be robust, tolerant and adaptable.

7.6.1 Methodology

The controller from section 7.5.1, using the mean, is set back into a four-way junction, except this time traffic will come from one road randomly chosen at the start of each test. Six cars, with a three second interval, was chosen as the test strategy (for a single road) as it exists as a good compromise between being quick to execute and being long enough to properly test controller operation. The three second interval, was determined through investigation, as it provides a good range of tests from sparse to queueing traffic. The test is conducted 50 times.

7.6.2 Results

The results from the tests presented in section 7.6.1 can be seen in figure 7.6. It shows that the epiNet controller has managed to improve with respect to the round-robin (previous testing of epiNet only showed marginal improvement in the order of seconds for balanced traffic). Figure 7.3 shows the breakdown of the duration of each car in each test conducted. It has been included to show that the first two cars spawned into the simulation have almost identical durations between simulations.

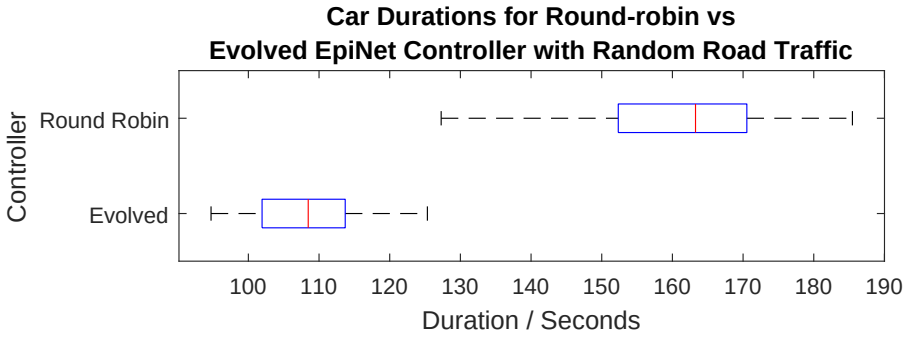

Figure 7.6: Box plot showing how long, in seconds (lower is better), it takes for cars to pass through the four way junction for Round-robin, and the evolved epiNet, controllers when traffic is present on only a single road (chosen at random). The graph shows that the evolved epiNet controller significantly outperforms the round-robin under these conditions.

It appears epiNet is good at adapting to new traffic behaviours it has never seen. However, it is important to note that the improvements epiNet enjoys cannot be categorically stated as owing to its adaptable, robust design. It could just be because the test environment is now better suited to the epiNet controller (which reacts to traffic) over the round robin (which would behave better in previous environments where traffic has been symmetrical and balanced). It is difficult to negate these conditions as there exists no scenario on the current junction in which the efficiency of the round-robin could be improved (therefore the evolved controller's improvements can never be ascertained as not being the result of the round-robin's

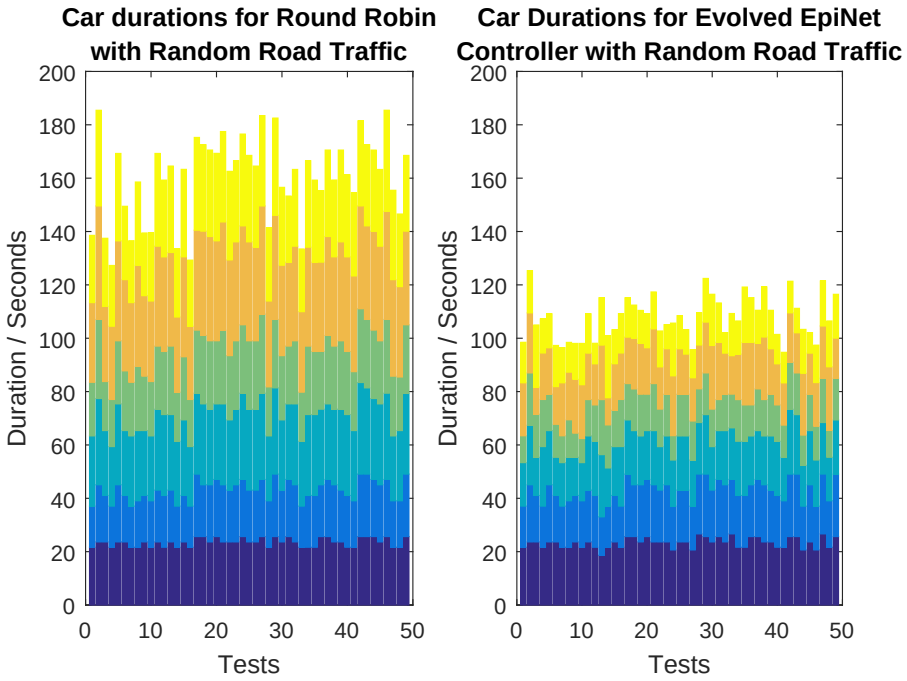

Figure 7.7: Stacked bar charts showing how long, in seconds (lower is better), it takes for cars to pass through the four way junction for Round-robin, and the evolved epiNet, controllers when traffic is present on only a single road (chosen at random). Each stack (shown in different colours) on each bar represents a different car's duration through the simulation. It can be seen that, while the evolved epiNet controller allows traffic to pass through it quicker in general, the first two cars consistently have similar durations between the two controllers.

degradation). Therefore the only conclusion that can be formed from this behaviour is verification that epiNet does react to traffic; it has not learnt a pattern or round-robin type system. The similarity between durations of the first two cars consistently between both controllers was unexpected. Through inspection of the simulation it was determined that the epiNet controller takes time to initially react to the stimuli of the first set of waiting cars. This latency is due the differences between how often both controllers update as explained in section 5.3.2.

7.7 Application Tolerance

This section investigates how well an epiNet controller, developed using a GA with a mean fitness averaging scheme, handles being used as a controller with different inputs and outputs from those it was evolved to use.

7.7.1 Methodology

To begin a new junction design is required. Removing road connections to any fewer (than three) would produce too simple a structure. Adding any more road connections (more than four) would result in too many controller inputs and outputs to be required for the previously evolved four-way junction controller. A three way junction was therefore decided upon. This junction is almost identical to the four-way junction except a location and corresponding road have been removed. Testing is also identical except, again, one set of traffic (for the non-existent location) is removed. The test was run three times: once using a round-robin controller; once with a three-way junction controller (that has been evolved); and once with a four-way junction controller.

7.7.2 Results

The results from the tests presented in section 7.7.1 can be seen in figure 7.8. It shows that round robin was the best controller method for a three-way junction. The salient finding is that the re-purposed four-way junction controller, whilst being the worse controller tested, was still capable.

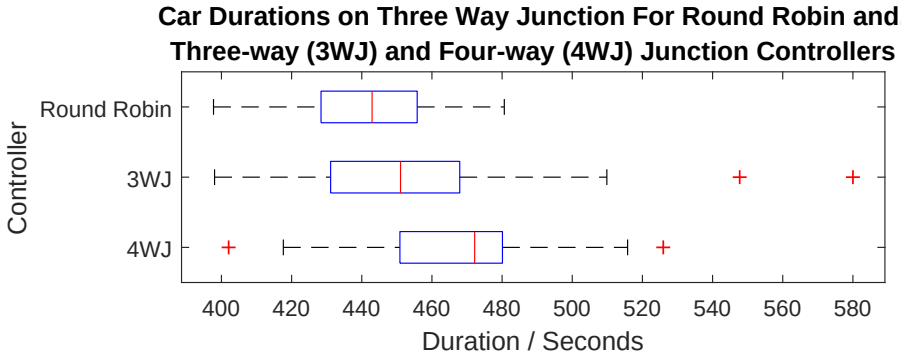

Figure 7.8: Box plot showing how long, in seconds (lower is better), it takes for cars to pass through the three way junction for: a Round-robin controller; an epiNet controller that was evolved on the three way junction (3WJ); and an epiNet controller that was evolved on a four way junction (4WJ). The graph shows similar times between all three controllers with the round-robin being the best on average and the 4WJ being the worse.

It is not surprising that the round robin was the best controller; the junction is incredibly simple (there isn't that many roads for the controller to cycle through) and the testing was symmetrical favouring the round-robin controller. It is impressive how well the four-way controller preformed considering it has no knowledge of the junction, the IO configuration or how to behave if inputs are missing. Again, this behaviour is not surprising; Turner in [20] notes how robust, tolerant and adaptable these types of system can be.

7.8 Summary

In this chapter a methodology of how to design and evolve a single epiNet-based traffic controller, that can rival a round robin scheme, has been successfully developed. Three main requirements were determined:

- Evolution requires a Genetic Algorithm.
- Controller inputs should be individually mapped to whether a road has a car waiting, and how long it has waited. Controller outputs should be binary encoded as to which road should next be activated.
- The Genetic Algorithm used should use the mean to average test results within its fitness function.

It was also found that epiNet controllers react to traffic stimuli as supposed to learning patterns. This makes them better at handling unbalanced traffic over a round-robin controller. Additionally they were found to be robust enough to handle being applied to controllers which require less inputs with no prior experience. A major drawback found with epiNet was the time taken for it to react to changes in its inputs. However, this latency was artificially implemented at one second (in an attempt to mimic real world latency) and it could be argued that this could be over zealous and should therefore be reduced mitigating this drawback. This chapter does not consider this however, as it does not affect the intention of this chapter (the methodology will be unaffected by a reduction in latency). Therefore this point is not considered further at this stage.

The next chapter build upon the findings of this section to develop a methodology which uses epiNet to control a network of traffic controllers.

Chapter 8

Evolving a Network of EpiNet Traffic Controllers

Contents

In chapter 7, a methodology for evolving a single controller was developed. This chapter looks to improve upon this methodology as it investigates how to evolve epiNet to handle a network of traffic controllers. Inspiration will be taken from biology in this regard as principles of cooperative evolution and hierarchical structures (described in chapter 6) are applied. The chapter will conclude with a summary of the key findings from these investigations.

8.1 Network of Four Four-way Junctions

Tests conducted in this chapter refer to a network of four way junctions. This section therefore seeks to introduce this road network and how it behaves within the agent-based simulation. First consider the screen shot of it given in figure 8.1. There exists four, four-way junctions

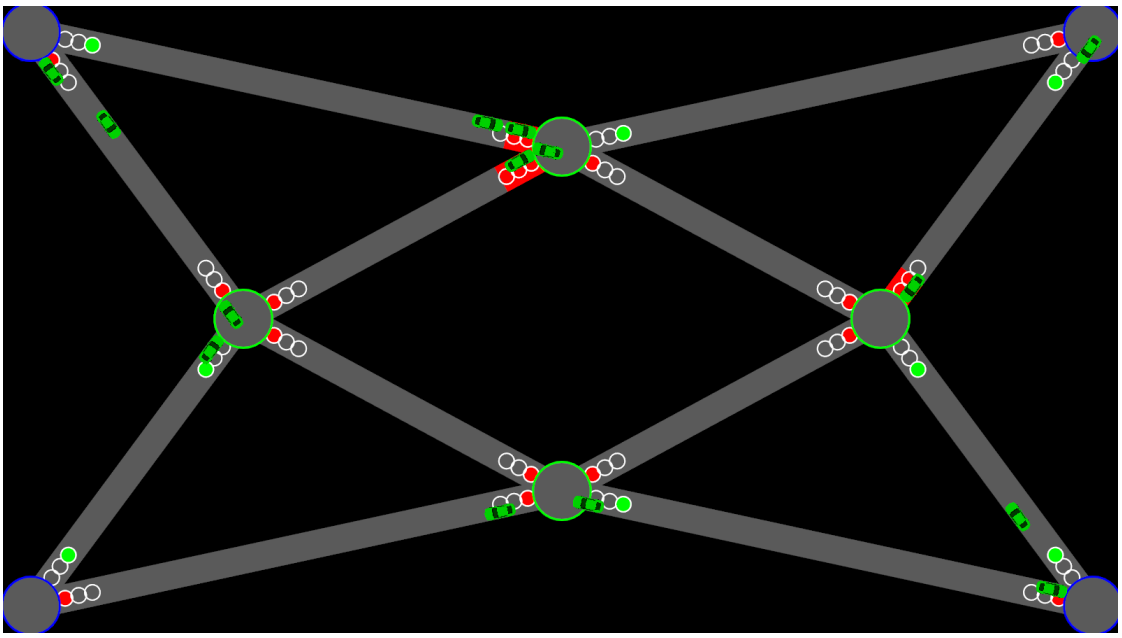

Figure 8.1: Screenshot from the agent-based simulation's GUI as it displays the four, four-way junction (also referred to simply as the traffic network).

in the centre of the network. Two road connections from each junction extend towards separate locations (that spawn/ destroy cars) at the edge of the network. The two other road connections extend towards other four-way junctions. The design was chosen as it allows for complex traffic tests to be designed. For instance the following can be easily chosen at random: a major road with heavy traffic with two intersecting minor roads; or all major roads with fluncuating heaving and light traffic; or two independent major roads; and so on. Four-way junctions were chosen such that controllers developed in chapter 8 can be reused. It is difficult to predict what controller behaviour will yield the most efficient control network (especially with view of different traffic directions and densities). This is ideal, one of the major goals of this research is to investigate whether epiNet controllers can find efficient control schemes which are non-obvious or complex.

8.2 Imported Four-way Junction Controller

The advantage of using four, four-way junctions is that the controller developed in chapter 8 can be cloned for each control system. This will allow observation of how well it compares to a round-robin in a traffic network. It will also provide a good comparison to assess the effectiveness of other strategies that evolve epiNet with knowledge of this traffic system. This section therefore presents how such testing was conducted and what results were found.

8.2.1 Methodology

The epiNet controller, designed in chapter 8, is required. Four separate instances of it are created. Each system takes four inputs and gives two outputs, as before, from a corresponding controller. Each controller is also still updated (together) every 100 frames. The testing scheme was chosen to be symmetrical (equal traffic distribution across the entire network) while testing complex behaviour (unbalanced traffic on each node). To understand this consider the traffic flow diagram show in figure 8.2. Each line of traffic represents three cars with a random interval of up to three seconds between them. This value was chosen through experimentation as it provided a good range of random traffic conditions between junctions (from sparse to queueing traffic) while remaining relatively quick to execute. This testing scheme was required to necessitate the emergence of complex traffic control requirements which needs to exist if the controllers are to develop complex behaviour.

8.2.2 Results

The results found from the tests presented in section 8.2.1 can be seen in figure 8.3. It shows that an epiNet controller developed on a single four way junction, where it was marginally better than a round-robin controller, also enjoys greater performance in a more complex setting.

This result is expected. Not only has it already be proven that the epiNet controller can outperform the round-robin, but as seen in section 7.7, it also excels within unbalanced traffic environments.

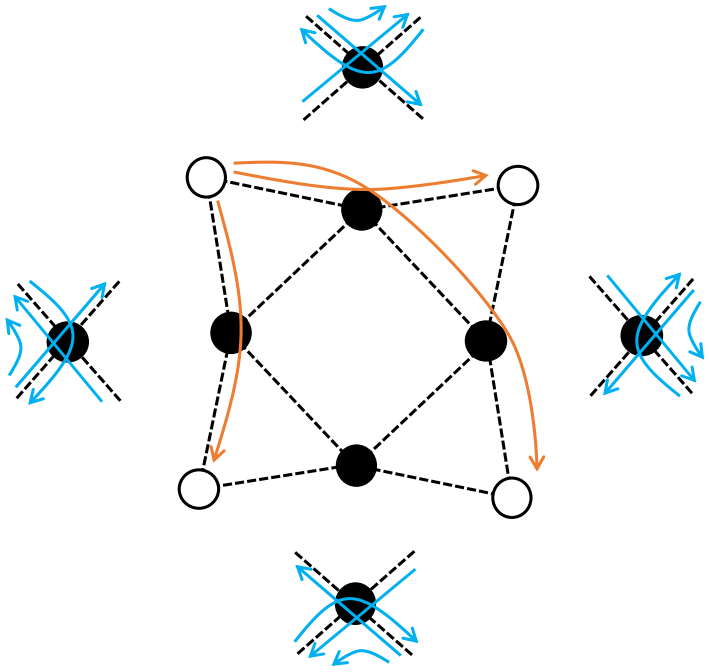

Figure 8.2: Diagram showing a test used on the traffic network of four, four-way junctions. The solid black nodes represent traffic systems (four-way junctions) whereas nodes with only a black outline represent locations (nodes that spawn/ destroy traffic). Dotted lines show the connecting roads between nodes. The centre portion of the diagram shows the aforementioned traffic network. The red lines shows the direction of traffic emanating from a single node; two lines of traffic take the shortest path to their neighbour locations and a further line extends clockwise around the network to the furthest network. This behaviour, though not shown explicitly, is repeated for each location. The smaller sub diagrams around the diagram show the traffic flow through the system they are adjacent to. The blue lines show all the traffic directions that pass through that system. The key point being that between all nodes, assuming controller mapping is consistent, the node collective experiences unbiased traffic. However, each individual node experiences biased traffic: a main road (2 lines); two minor roads (1 line); and an unpopulated road.

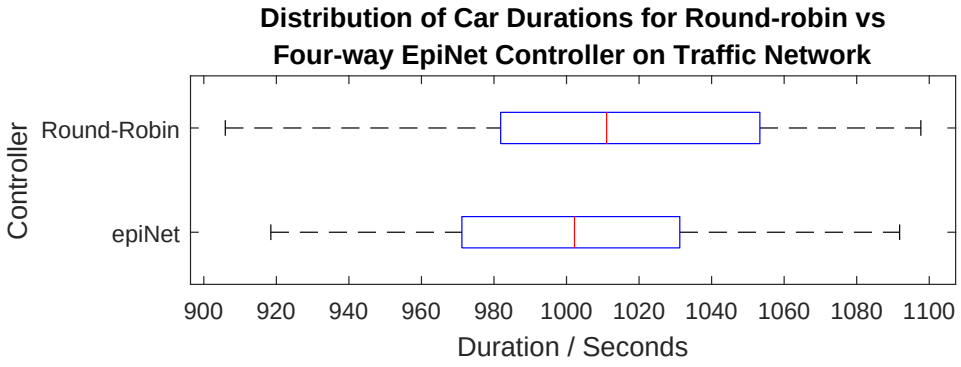

Figure 8.3: Box plot showing how long, in seconds (lower is better), it takes for cars to pass through the traffic network for round-robin controllers and epiNet controllers (which were evolved on a four-way junction and then duplicated onto the traffic network). The graph shows the epiNet controllers performed marginally better (smaller mean and spread).

8.3 Duplicate Controllers

In section 8.2 it was noted how by using a symmetrical traffic network of four, four-way junctions that the same controller can be used for each sub-system. This sections details how a

single controller was evolved on the network by duplicating it into each of the four sub-systems.

8.3.1 Methodology

The GA (used thus far) had to be altered such that it understands that, if specified by the user (by passing the evolve function the *duplicate* command), that it is required to duplicate the controller passed to it. Each controller is then handled, using standard map files, as normal in the simulation (detailed in chapter 5). Once evolved the methodology is unchanged from the one presented in section 8.2.1.

8.3.2 Results

The results, from the test descibed in section 8.3.1, show that duplicating controllers is an effective methodology.

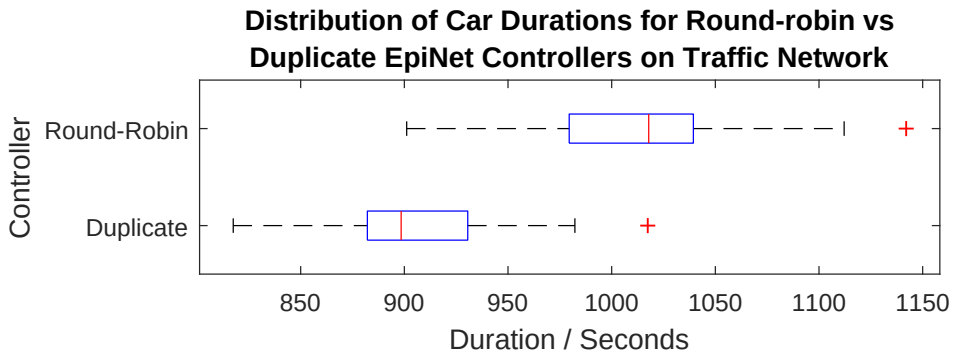

Figure 8.4: Box plot showing how long, in seconds (lower is better), it takes for cars to pass through the traffic network for the round-robin and duplicate epiNet controllers. The graph shows the duplicate epiNet controllers outperformed the round-robin controller.

This approach is likely so effective because in each fitness test (within the GA) the controller is exposed not only to more traffic data, but simultaneous different IO mappings. This leads to a more robust evolutionary environment for a single controller. However, observation of the simulation (with evolved duplicate controllers) yields the finding of complex, emergent behaviour suggesting this method does more than to just evolve an independent controller robustly. It can been seen that controllers no longer always switch to the next waiting car (as the best evolved single controller did in previous testing). Instead the controllers have learnt to package cars together on less busy roads and give the major roads preference. This, through observation, seems to make the single sub-system behave less preferably. With longer

viewing however, it becomes apparent that packaging cars together creates a rudimentary green wave (as discussed in section 4.1.3) meaning subsequent systems visited by the packets of cars spend much less time servicing that road. The resultant effect is a reduction in the duration for all cars to complete the simulation. However, this methodology is limited as it requires all controllers to have the same inputs and outputs. Additionally the test created is symmetrical, and it may be found that in asymmetrical arrangements the controllers (which will then have different needs) don't evolve to behave as preferably.

8.4 Single Controller

In nature, as seen in section 6.1.2, complex structures are used to control multiple separate processes and structures within a single organism. This section seeks to investigate whether this behaviour can be captured using a single large instance of epiNet to control all four junction controllers.

8.4.1 Methodology

A GA is used to evolve a single instance of epiNet that has sixteen inputs and eight outputs. From these IOs, four inputs and two outputs are sequentially mapped to the four controllers. The controllers are consistently mapped to the same inputs and outputs for all testing. The testing conducted within the GA evolutionary loop, and to test the fitness against the round robin post-evolution, is unchanged from the methodology presented in section 8.2.1.

8.4.2 Results

It can be seen in figure 8.5 that, even after five hundred generations, that the single epiNet controller could not compete with the round-robin. Figure 8.6 shows how the controller converged upon this solution very quickly and is unlikely to ever improve its performance given a longer evolutionary period.

The fact this methodology converged upon a solution within fifty generations is surprising. In section 7.4 it was demonstrated how epiNet needed simple IOs to assist its process of converging upon a solution (and thus evolving). This convergence seems to defy this previous

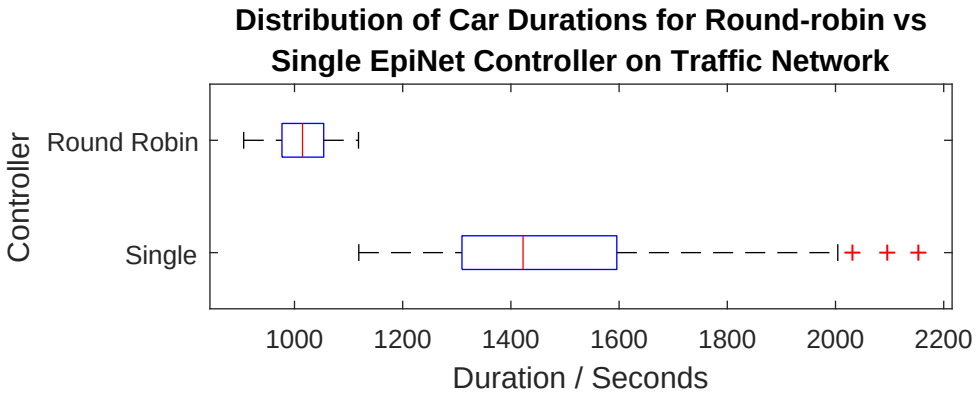

Figure 8.5: Box plot showing how long, in seconds (lower is better), it takes for cars to pass through the traffic network for the round-robin and a single epiNet controllers. The graph shows the round-robin controller significantly outperformed the single epiNet controller.

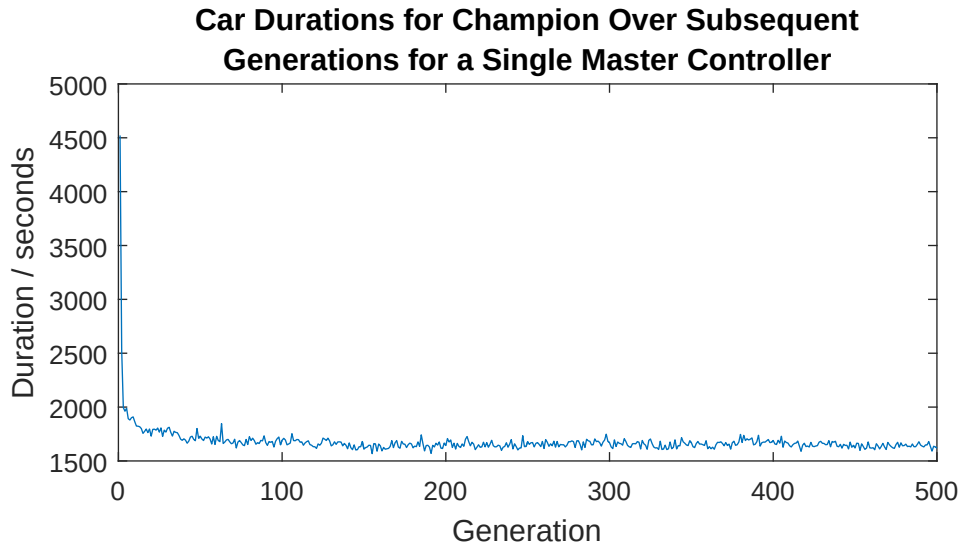

Figure 8.6: Graph showing how long, in seconds (lower is better), it takes for cars to pass through the traffic network for the fittest single epiNet controller in the each generation of the GA. It can be seen the that the initial duration (of over 4500 seconds) rapidly converges onto its final solution. The noise on this converged value is likely due to the stochastic nature of the tests conducted.

finding. This could have occurred through chance (this case just happened to find genetic improvements) or maybe both fitness landscapes are sufficiently different that, while both become stuck at local optimums, there exists at least some room for initial convergence within the more complex traffic structure. The latter would suggest that greater evolutionary technique (a more evolveable GA for example) may later provide a breakthrough for a hierarchical control structure. Without more testing, however, it is impossible to draw any definitive conclusions in this regard. At this stage hierarchy has not been found to be able to compete with a round robin in traffic control.

8.5 Separate Controllers

In section 6.1.1, it was seen how competing organisms may evolve together as they apply evolutionary pressure upon one another. This section seeks to investigate whether this behaviour can be captured by evolving multiple instances of epiNet to control separate networks individually.

8.5.1 Methodology

The first critical point to make is that the methodology for this section does not follow cooperative evolution as defined by Potter and Jong in [41]. Though this approach was considered, the computational time to conduct such a methodology exceeded that which was feasible; it would require the number of simulations (currently taking approximately a week) to increase by a power of four. Instead a simpler approach was decided upon. The first step was to modify the GA (developed thus far) such that each individual in a population becomes a set (or more accurately a vector) of controllers, not just a single controller. This means that the general evolutionary process (first introduced in section 7.1) crossovers and mutates a set of controllers, not just a single one. Crossover and mutations are only allowed between the same controller index between individuals such that separate populations are preserved. Each set of epiNet controllers (each individual) is then applied to test simulations, and the GA, as normal. The drawback of this methodology is that finding a *best* set relies heavily on random chance (as supposed to explicitly testing every combination of controllers between sub-populations as seen in cooperative evolution [41]). Ultimately this means the implemented methodology will be prone to becoming stuck at local optimums if the improvement of one controller relies on the degradation of another (and that degredation is more costly than the improvement is valuable). The testing conducted within the GA evolutionary loop, and to test the fitness against the round robin post-evolution, is unchanged from the methodology presented in section 8.2.1.

8.5.2 Results

It can be seen in figure 8.7 that, even after five hundred generations, that separate epiNet controllers could not compete with the round-robin. Figure 8.8 shows how the controllers

initially converged quickly toward a better solution, but it is not clear whether (as seen by applying a moving average) a final value has been converged upon.

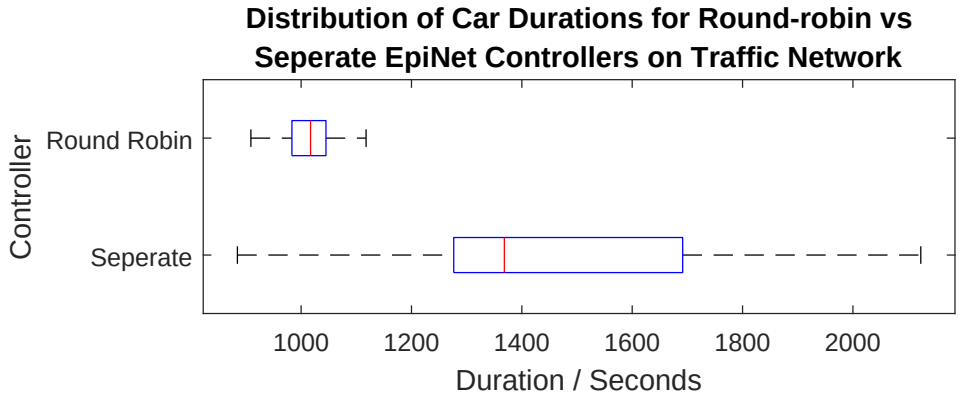

Figure 8.7: Box plot showing how long, in seconds (lower is better), it takes for cars to pass through the traffic network for the round-robin and separate epiNet controllers. The graph shows the round-robin controller significantly outperformed separate epiNet controllers.

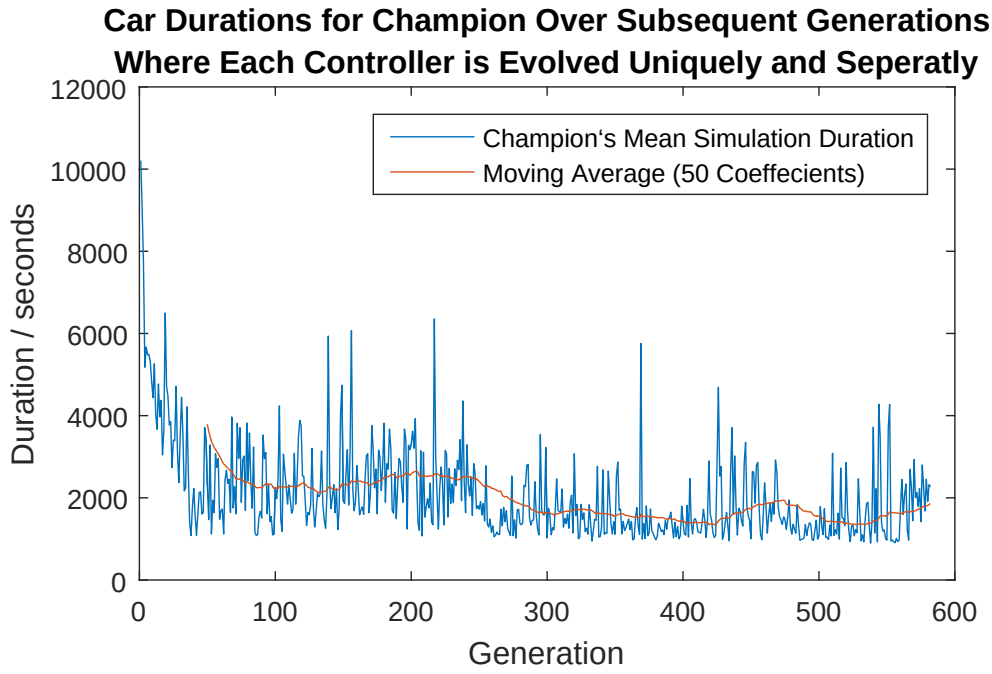

Figure 8.8: Graph showing how long, in seconds (lower is better), it takes for cars to pass through the traffic network for the fittest set of separate epiNet controllers in the each generation of the GA (*blue line*). It can be seen the that the initial duration (of over 10000 seconds) slowly converges to a smaller value and that this convergence is very noisy. By looking at the moving average (*red line*) it is unclear whether the controllers have converged upon a solution.

It was expected that by not implementing the complexity mandated by the cooperative evolution methodology that poor performance would result. This is because the traffic application has already been established as having a complex search space. The evolutionary progress data is interesting however. Most notably the waveform is extremely noisy (especially compared to other evolution patterns such as figure 8.6). This is likely result of the random process in

which better sets of individuals are chosen (adding random noise to the evolutionary process). Additionally the controllers may have not converged upon a solution. They can converge quickly under the correct circumstances (as seen by the first fifty generations of evolution) and it was anticipated that they may become stuck at a local optimum quickly (as with other strategies such as 8.6). But by looking at the moving average, it is unclear whether this has been the case. Though not feasible, it would be interesting to observe whether over significantly more generations whether this method continues to successfully navigate the search space or whether it converges. Ultimately, however, it can only be concluded at this stage that having unique controllers is infeasible due either to computational complexity of evolving them (in the case of the cooperative evolution methodology) or ineffectiveness (considering the implemented methodology).

8.6 Summary

Many different methodologies that seek to develop controllers that minimise the duration of cars through a simulated traffic network (of four, four-way junctions) have been investigated. The findings are combined in figure 8.9. The symmetrical nature of the design lends itself

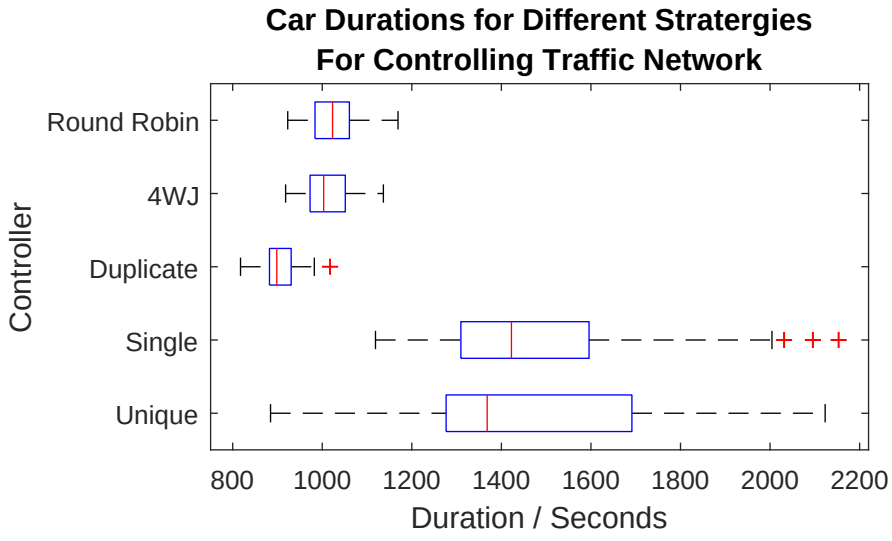

Figure 8.9: Box plot showing how long, in seconds (lower is better), it takes for cars to pass through the traffic network for: round-robin controllers; epiNet controllers evolved on the four-way junction (4WJ); duplicate epiNet controllers; a single epiNet controller; and separate (unique) controllers. The graph shows the duplicate controllers performed the best on average. The round-robin and 4WJ controllers were similar (the 4WJ being marginally better). And the single and unique controllers performed poorly.

toward the round-robin, single four way junction duplicated (4WJ), and the evolved duplicate

controllers. The latter two of which are infeasible if the inputs and outputs of each system (with the traffic network) aren't consistent (or compatible). The single controller performed poorly owing to its complex IO scheme that, as previously found, is too difficult to evolve. A more proficient EA may remedy this situation. Evolving unique controllers requires a formal co-evolution methodology (such as [41]) and significantly greater computational resources to ascertain whether it is a feasible strategy. It is therefore the finding of this section, that without further investigation, that if the traffic network is symmetrical, with common traffic systems, that evolving duplicate controllers is the most effective strategy. This is because it allows the emergence of complex behaviour owing to the increased test behaviour it has access to. If the traffic network is inconsistent, or controllers are unable to be duplicated effectively, evolving controllers separately on custom, individual scenarios is the best strategy. This allows for a robust controller. However the requirement of this second methodology is questionable, as it could be argued that a controller could be explicitly designed for a single junction more effectively that could be evolved using epiNet.

Chapter 9

Conclusions and Further Work

Contents

This chapter summarises the project presented in this report. This begins by first recapping what work has been conducted before summarising the major findings that arose from it. These findings are accompanied with discussion into their validity, any limiting factors and ultimately what knowledge they have contributed. Potential future avenues are then explored before final concluding remarks are given.

9.1 Work Conducted

The project was primarily motivated by a desire to investigate the effectiveness of using different evolutionary techniques in optimising networks of epiNet controllers. To facilitate this an application was required. Traffic control was chosen as, not only is it an ideal research environment, but simulations can be graphically observed which is useful for demonstration

and visual inspection. Additionally congestion is a growing, costly problem and thus tackling this is also worthwhile.

To facilitate this research three major elements needed developing: an agent-based traffic simulation; an epiNet controller (ported to C++); and an evolutionary algorithm. The agent-based simulation was designed to be computationally efficient, such that it could be run multiple times quickly, and adaptable so that new tests could be conducted easily without having to manually re-write large sections of code. EpiNet had to be ported to C++ as this was desired over the less workable alternative of using DLLs. This program then required integration with the simulation such that it could adequately control the lights of a traffic junction. Finally an evolutionary algorithm was developed and applied to the epiNet controller. The methodology for doing this was developed over multiple investigations. The conclusions drawn from these investigations form the research findings of this project.

9.2 Findings

9.2.1 Single Controller

The first set of investigations sought to develop a methodology for designing a single controller. They found an effective controller, that could outperform a round-robin configuration and demonstrate both robustness (handle new unseen traffic patterns effectively) and adaptability (can effectively take control of similar smaller junctions), could be developed if the following points were abided by:

1. A genetic algorithm is used to evolve the traffic controller.
2. The genetic algorithm uses the mean to average multiple fitness tests (controller should be tested on multiple randomly-seeded simulations).
3. Traffic controllers have simple inputs and outputs.

The first and last points unfortunately are vague due to the fact that limited testing was conducted for each. A single genetic algorithm was chosen, and its exact methodology was picked somewhat arbitrarily. It is therefore impossible to ascertain from this investigation alone exactly which GAs are required. Only three IO configurations were formally investigated. While it was seen that by reducing the IOs to the smallest number (still capable

of interpreting traffic) yielded the best results it has not been ascertained what number of inputs and outputs lead to an ineffective controller. Therefore the best conclusion that can be drawn for these two points is that the GA defined in section 7.3 can effectively evolve a controller that has up to 4 inputs and 2 outputs using the mapping scheme described in section 7.4.

9.2.2 Networking Controllers

The second set of investigations sought to expand the single controller methodology to use epiNet to control a network of traffic controllers. They were four salient findings arising from these tests:

1. Evolving a single epiNet controller (inspired by hierarchy in biology) to control multiple control systems is infeasible as the inputs and outputs will likely be too complex to evolve.

2. Co-evolving multiple epiNet controllers to control multiple control systems is infeasible as the evolution time required will likely be too great to be computationally feasible.

3. Evolving a single epiNet controller and duplicating it between traffic systems in a symmetrical, uniformly-sized traffic network is an effective strategy that leads to the emergence of complex behaviour.

4. Evolving a single epiNet controller on a separate, custom simulation and then applying it to a complex traffic system (multiple different controllers can be separately evolved) is an effective strategy for asymmetrical, or inconsistently-sized, traffic networks.

The first two points are not conclusive. Investigations suggested that hierarchy may still be viable if more powerful evolutionary techniques are implemented to handle the increased search space complexity arising from a greater number of inputs and outputs. Co-evolution may also be viable if greater computational resources can be allocated. Both methodologies would also benefit if applied to smaller traffic networks (with only two or three junctions for instance). Both methods will never be able to scale well however. The third point, while showing general promise, is unlikely to be effective within traffic control design. This is because traffic networks are typically not symmetrical and road junctions within them tend to vary radically (and hence a situation where this can be used is likely to be rare). This methodology may be useful however in other applications which are inherently more symmetrical. The

fourth point is also unlikely to be implemented as it essentially looks to evolve a controller in a reduced environment. Because of this it cannot learn nuances about the system it will eventually control, and thus never demonstrate complex behaviour. This is a poor choice as more efficient controllers can be designed using classical control methods.

9.3 Hypothesis Revisited

The hypothesis of this report stated that:

> Implementing epiNet as an adaptive traffic controller reduces congestion within a traffic network as it inherits useful biological traits including robustness, adaptability and fault tolerance.

The work conducted in this report led to a network of epiNet-based traffic controllers being developed which reduced the duration cars took to pass through the simulation. It was able to achieve this over simulations whose traffic density randomly varied. As such it can be inferred that congestion must therefore have been reduced. However, by only testing a single objective (reducing the total duration cars took to complete their routes within the simulation) it cannot be definitively concluded that traffic performance is universally improved. For instance, the data provides no indication if cars are more commonly being stopped (which wastes energy and thus greater emissions can be expected as cars do more work through acceleration). The work conducted to evolve a single controller, in chapter 7, supports the claim that the controller achieved these advantages by virtue of its robust and adaptable traits; the controller was able to handle unseen traffic patterns and adapt to handle strange IO configurations.

The hypothesis also stated:

> Applying a genetic algorithm to evolve a single instance of epiNet in a network limits the performance of the program compared to using a method that allows individuals to apply evolutionary pressure upon one another.

It was seen for a single case (symmetrical traffic networks with common road junctions) that evolving duplicate controllers simultaneously led to a controller that could outperform a methodology that evolved controllers separately. The duplicated controllers achieved this by demonstrating that complex behaviour had emerged. This therefore supports the hypothesis. However, further work is required to completely validate this statement, as currently it only

holds true for a single case.

9.4 Further Work

The work conducted is this project explored many research avenues, with many being under-developed, as it reached toward a breakthrough. This section will briefly summarise these under-developed areas, already touched on in this report, before more deeply discussing two possible future directions in which research could be conducted.

9.4.1 Under-developed Research

The following points were not fully concluded over the course of the project:

- Which exact GA should be used to evolve epiNet with view of the complex traffic environment?
- What is the limiting complexity of an epiNet-based controller's IOs?
- How does changing the simulation objective (such as reducing the total waiting time of cars) change the performance of an evolved controller?
- How does changing the rate at which epiNet is updated within the simulation change controller performance?

9.4.2 Neighbourhoods

Currently no methodology has been discovered into how best to evolve controllers, using epiNet, for asymmetrical networks with junctions that have differing numbers of connected roads. Two methods have already been considered: cooperative evolution and hierarchy.

With regard to cooperative evolution it became apparent that the formal methodology (shown in [41]) is required to be implemented. However it was too computationally expensive to implement. This could be rectified if greater computational resources could be sourced, smaller networks are considered and the speed of each test could be reduced. However, this approach

will always have a scaling issue and as such this methodology requires ingenuity in resolving this if it is to be worth pursuing. One avenue that may be worth considering is the use of neighbourhoods, where only local or heavily-dependent controllers are considered together, to prevent the need to include every epiNet instance in the process.

Hierarchy was discounted because of its complex IO scheme. However, with a more powerful GA and a smaller network application this could be remedied. This approach will ultimately struggle with, like cooperative evolution, scalability. Research could therefore be conducted into whether hierarchy could be used in a supportive role, as supposed to being the sole controller, such that it need not consider as many inputs and outputs. Naturally it seems to follow that these supportive structure cannot encompass the entire network, as unavoidably, they will become too complex (too many IOs) to evolve. Therefore, again, using neighbourhoods of controllers with a hierarchical master-controller supervising them seems more viable.

9.4.3 Universal Adaptive Traffic Controller

Evolving a single controller was relatively easy. However, its performance showed no signs of complex behaviour (and hence its usefulness as an actual control left questionable). This arose from the fact the environment it was evolved in was also simple having only a single, simple objective and a relatively small amount of training data. An interesting avenue of research may be to improve this evolutionary environment mainly through the use of Multi-Objective (MO) design and better simulating the requirements of the intended junction. For instance pollution, fuel consumption, noise and vehicle wear could each be reduced while increasing road capacity and journey times. Controllers could also be rewarded for the quality of cars leaving the junction (other junctions prefer traffic packeted together). The exact densities of traffic upon different roads (and how they vary over time) could also be more accurately modelled to allow epiNet to develop dynamic control bias.

9.5 Final Remarks

The project has set solid foundations for valuable, future research opportunities. Although there have been no revolutionary breakthroughs yet found, it is credible to expect that epiNet could, assuming future research, be incredibly effective as a traffic controller. Furthermore

this could, by extension, impact a wide-range of other similar applications. It is therefore genuinely hoped that this research is continued looking forward.

Bibliography

[1] A.P. Turner, M.A. Trefzer, and A. Tyrrell, "Modelling epigenetic mechanisms to capture dynamical topological morphology : Applications in edge detection," 2015.

[2] A.P. Turner, M.A. Lones, L.A. Fuente, S. Stepney, L.S.D. Caves, and A.Tyrrell, "The artificial epigenetic network," in *SSCI13 2013, Singapore, April 2013*, IEEE Press, 2013, pp. 66–72.

[3] INRIX (2014, Oct. 14). "Traffic Congestion to Cost the UK Economy More Than 300 Billion Over the Next 16 Years," inrix.com. [Online]. Available: inrix.com/press/traffic-congestion-to-cost-the-uk-economy-more-than-300-billion-over-the-next-16-years/ [Accessed: Nov. 14, 2015].

[4] W. Xin, J. Chang, S. Muthuswamy, and M. Talas, "Midtown in motion: A new active traffic management methodology and its implementation in new york city," in *Transportation Research Board 92nd Annual Meeting*, 2013.

[5] D.M. Nicol, "Hacking the lights out," *Scientific American*, vol. 305, no. 1, pp. 70–75, 2011.

[6] J.M. Bower and H. Bolouri, *Computational Modeling of Genetic and Biochemical Networks*, ser. Bradford Books. Cambridge: MIT Press, 2001.

[7] K. Chao, R. Lee, and M. Wang, "An intelligent traffic light control based on extension neural network," in *Knowledge-Based Intelligent Information and Engineering Systems*, ser. Lecture Notes in Computer Science, I. Lovrek, R.J. Howlett, and L.C. Jain, Eds., vol. 5177, Berlin: Springer Heidelberg, 2008, pp. 17–24.

[8] X. Yu and W. Recker, "Stochastic adaptive control model for traffic signal systems," *Transportation Research Part C: Emerging Technologies*, vol. 14, no. 4, pp. 263–282, 2006.

[9] G. Walsh, *Proteins: Biochemistry and Biotechnology*. Chichester: Wiley, 2002.

[10] J. W. Pitera and W. Swope, "Understanding folding and design: Replica-exchange simulations of"trp-cage" miniproteins," *Proceedings of the National Academy of Sciences*, vol. 100, no. 13, pp. 7587–7592, 2003.

[11] G.A. Petsko and D. Ringe, *Protein Structure and Function*, ser. Primers in biology. London: New Science Press, 2004.

[12] V.A. Bloomfield, D.M. Crothers, and I. Tinoco, *Nucleic Acids: Structures, Properties, and Functions*. California: University Science Books, 2000.

[13] B. Alberts, A. Johnson, J. Lewis, D. Morgan, M. Raff, K. Roberts, and P. Walter, *Molecular Biology of the Cell*. Abingdon: Taylor & Francis Group, 2014.

[14] F. H. C. Crick and J. D. Watson, "The structure of dna," *Nature*, vol. 171, no. 4356, pp. 737–738, 1953.

[15] M. A. Tribe, *Protein Synthesis*, ser. Basic Biology Course. Cambridge: Cambridge University Press, 1976.

[16] S. Malcolm, J. Goodship, and T.H.J. Goodship, *Genotype to Phenotype*, ser. Human molecular genetics series. Abingdon: BIOS Scientific, 2001.

[17] N. M. Gericke and M. Hagberg, "Definition of historical models of gene function and their relation to students understanding of genetics," *Science & Education*, vol. 16, no. 7-8, pp. 849–881, 2007.

[18] D. Latchman, *Gene Regulation*, ser. Advanced Texts. Abingdon: Taylor & Francis, 2007.

[19] F. Jacob and J. Monod, "Genetic regulatory mechanisms in the synthesis of proteins," *J. Mol. Biol.*, vol. 3, pp. 318–356, 1961.

[20] A.P. Turner, "The artificial epigenetic network," PhD thesis, Dept. Elect. Eng., The Univ. of York, York, 2013.

[21] J. Craig and N.C. Wong, *Epigenetics: A Reference Manual*. Norfolk: Caister Academic Press, 2011.

[22] R. Bonasio, S. Tu, and D. Reinberg, "Molecular signals of epigenetic states," *Science*, vol. 330, no. 6004, pp. 612–616, 2010.

[23] H. Hamann, T. Schmickl, and K. Crailsheim, "Coupled inverted pendulums: A benchmark for evolving decentral controllers in modular robotics," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ACM, 2011, pp. 195–202.

[24] A.A Puntambekar and I.A Dhotre, *Systems Programming*. Pune: Technical Publications, 2008.

[25] B. Stroustrup, *The C++ Programming Language*. Pearson Education India, 1986.

[26] *Traffic control system design for all purpose roads (compendium of examples) manual*, A, Department of Transport, Highways Agency, London, 2003.

[27] X. Xie, S.F. Smith, T.i Chen, and G.J. Barlow, "Real-time traffic control for sustainable urban living," in *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, IEEE, 2014, pp. 1863–1868.

[28] J. Barceló, *Fundamentals of Traffic Simulation*, ser. Int. Series in Operations Research & Management Sci. London: Springer, 2011.

[29] L. Gomila, "Simple and Fast Multimedia Library,". [Online]. SFML. Available: www.sfml-dev.org/. [Accessed: Apr. 20, 2016].

[30] L. Valente, A. Conci, and B. Feijó, "Real time gameloop models for single-player computer games," in *Proceedings of the IV Brazilian Symposium on Computer Games and Digital Entertainment*, vol. 89, 2005, p. 99.

[31] M. Cohn, *User Stories Applied: For Agile Software Development*, ser. Addison-Wesley Signature Series. Addison-Wesley, 2004, p. 3.

[32] X. Yu and M. Gen, *Introduction to Evolutionary Algorithms*, ser. Decision Engineering. London: Springer London, 2010.

[33] C. Darwin, *On the origin of species*. New York: D. Appleton and Co., 1871.

[34] R. Dawkins, *The Selfish Gene*. New York: Oxford University Press, 1976.

[35] C. Darwin, *On the Various Contrivances by which British and Foreign Orchids are Fertilised by Insects: And on the Good Effects of Intercrossing*. London: John Murray, 1862.

[36] J.N. Thompson, "The population biology of coevolution," *Researches on Population Ecology*, vol. 40, no. 1, pp. 159–166, 1998.

[37] T. F. H. Allen and T. B. Starr, "Hierarchy: Perspectives for ecological complexity.," *Behavioral Science*, vol. 28, no. 4, pp. 305–306, 1983.

[38] S.J. Gould, *The Structure of Evolutionary Theory*. Cambridge: Harvard University Press, 2002.

[39] T. Strachan and A.P. Read, *Human Molecular Genetics 3*. New Delhi: Garland Science, 2004.

[40] T. Bäck, A.B. Fogel, and Z. Michalewicz, *Evolutionary computation 1: Basic algorithms and operators*. CRC Press, 2000, vol. 1.

[41] M.A. Potter and K.A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel problem solving from naturePPSN III*, Springer, 1994, pp. 249–257.

Project Management

A formal project plan was implemented and followed to ensure development remained on-track to reach its desired goals. This chapter seeks to detail this plan by first outlining the exact methodology chosen. The resources required to complete the project are then listed before aims of the project are stated. The milestones are then presented with time management considered. Finally project risk is outlined.

Methodology

This project utilised an AGILE (iterative) approach to development summarised by weekly reports and an ongoing Gantt chart. This allowed for flexible, adaptive software development whilst facing the implementation of features who's difficulty of implementation was difficult to predict. Another advantage of iterative development is that working code was regularly produced. This allowed for time-consuming evolution tests to be run, on external machines, as early as possible in the project. Further software development could be conducted in parallel to this testing meaning time was not wasted waiting on experiments.

Resources

The resources used for this project were as follows:

1. Single PC running the Windows OS (Windows 7 or later).
2. Microsoft Visual Studio Community 2015 IDE.
3. Matlab IDE (2014a or later).

4. Bitbucket online repository.
5. Various digital storage (for the purposes of back-ups).
6. Gantter.
7. Access to journals, academic papers, and other educational materials.

Alternatives exist for some of the resources listed. These were chosen either because of availability or familiarity to the author.

Aims & Objectives

1. Evolve epiNet-based traffic controllers:
   (a) Define a standardised form for traffic controllers (which meet the mandatory traffic law requirements.)
   (b) Develop a simple agent-based traffic simulation for multiple controllers in C++.
   (c) Develop an EA (with fitness function using MOEs) to evolve a single controller.
   (d) Test whether evolved controllers improve the simulation MOEs over time.
2. Develop and apply evolutionary schemes to evolve epiNet-based traffic controllers on traffic network:
   (a) Modify the EA to develop a set of controllers based upon different evolutionary techniques.
   (b) Test, and thus determine, which, if any, technique produces the best controller network by comparing simulation results.

Note that two additional objectives were de-scoped over the course of the project due to time constraints. These were to: apply multi-objective design schemes to epiNet-based traffic controllers; and implement epiNet-based traffic controllers on professional simulators. These aims are formed into a work breakdown structure seen in figure A.1 in the appendices.

Milestones

The following builds upon the project's aims, presented in section , to introduce a formal set of project milestones (some being time sensitive):

1. Literature review completed (deadline: 26/11/16)
2. Phase 1 completed.
   (a) Traffic simulation created.
   (b) EpiNet integrated as controller within traffic simulation.
   (c) Methodology to evolve a controller for a single junction determined
3. Phase 2 completed.
   (a) Simulation adapted to handle multiple controllers.
   (b) Strategies to evolve controllers on traffic network evaluated.
4. Project presentation conducted (On: 13/04/16)
5. IAC project demonstration completed (On: 27/04/16)
6. Final report completed (deadline: 12/05/16)
7. Viva Voce (deadline: 19/05/16)

Time-line

The project time-line, based upon the requirements introduced in this chapter, is presented in figure 9.1 as a road map. The road map considers the entire project starting from when the literature review was began (starting 16/10/2015) to when the viva voce will be conducted (on 19/05/16). There are two main phases to the project. The first looks to construct all the necessary tools to facilitate research, and the second looks to conduct said research. There is a week between finishing the second phase and starting the project report to allow other work to be completed (namely writing the presentation). A more complete break-down of the activities completed is given by the gantt chart shown in figure A.2 in the appendices.

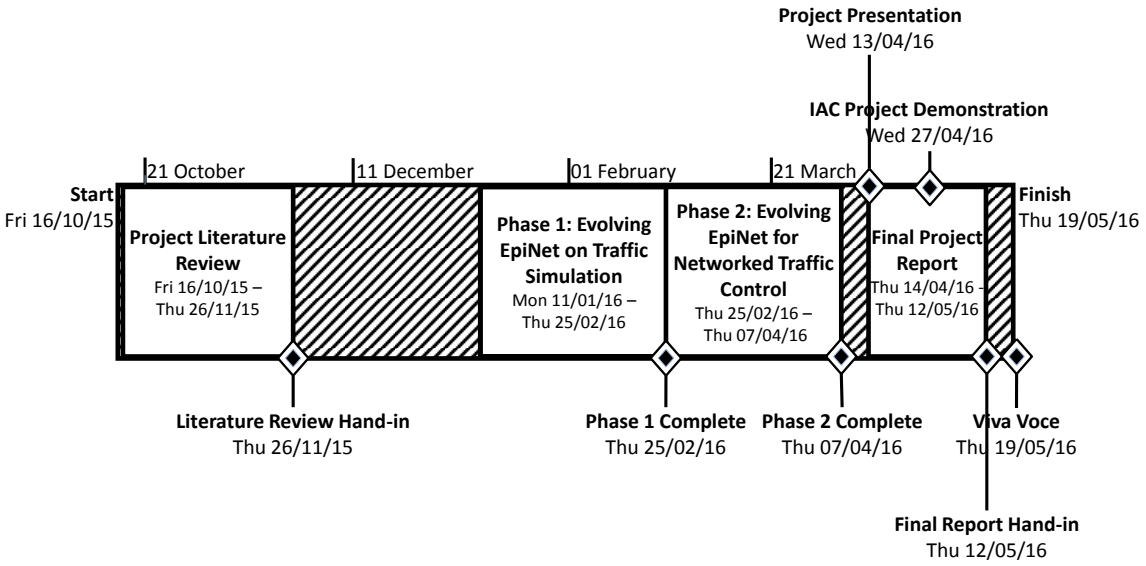

Figure 9.1: Roadmap for the project outlined in this report. The figure shows a timeline moving from the project start date (16/10/15) to its end (19/05/16). The four main bodies of work completed during the project are shown as blocks on the time-line. Milestones are also shown (as diamonds) indicating important deliverables in the project.

Comparison to Initial Plan

The initial time management plan is shown, via Gantt chart, in figure 9.2. Figure 9.3, which shows the final plan, can be directly compared to see how the plan changed over the course of the project. Considering the project chronologically, the first difference is that the project started a week later than expected. This was due to other commitments taking more time to complete than expected delaying development. The first phase took longer than initially anticipated (by two weeks). This was predicted early within the first phase, and hence a new milestone was made to track the progress of the phase (to track progress more concretely). This extension was due to epiNet needing to be ported, an activity which was not initially anticipated. The second phase was also extended as tests took longer to complete than expected and other commitments diverted more resources than anticipated away from the project. It was decided mid-way through this phase that the aspirational tasks (3a and 3b in figure 9.2) needed to be removed as there was insufficient time to complete them. By doing this the project was put back on schedule (by keeping the time to complete the report consistent).

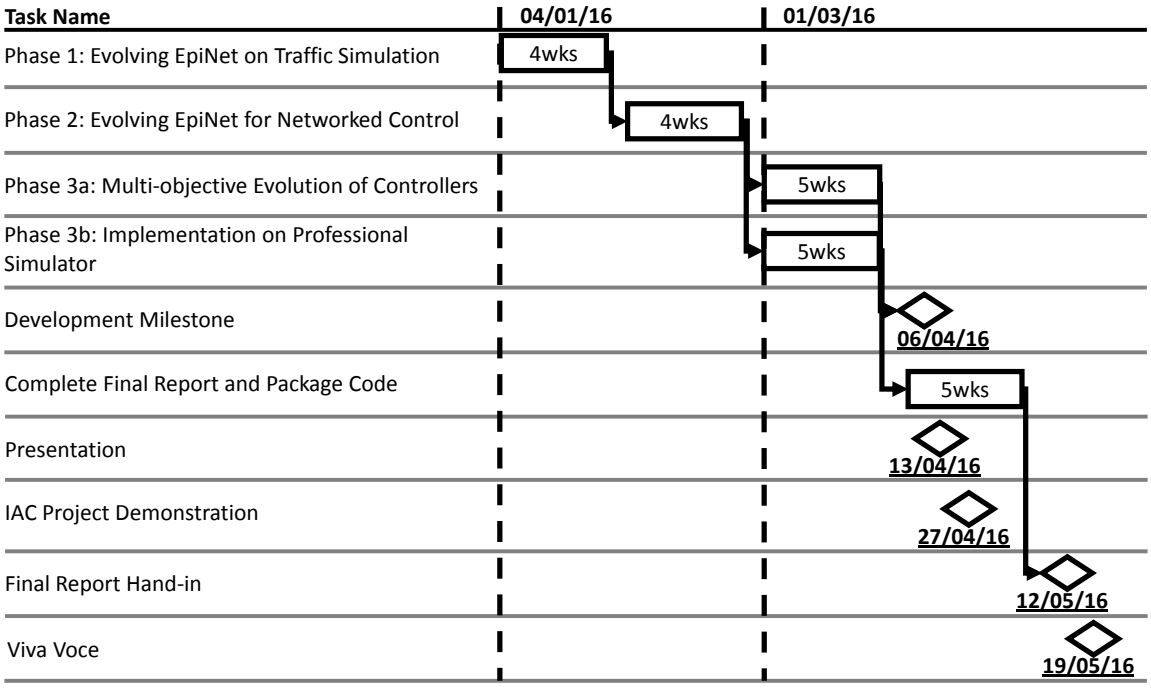

Figure 9.2: Gantt chart for the **initial** project plan. High-level tasks are listed at the left hand side. Blocks are used to indicate when each should being, and how long work is expected to take. Arrows show precedence between tasks (the pointed to task cannot start till the predecessor is complete. Diamonds show milestones for the project with their date of completion noted directly below them.)

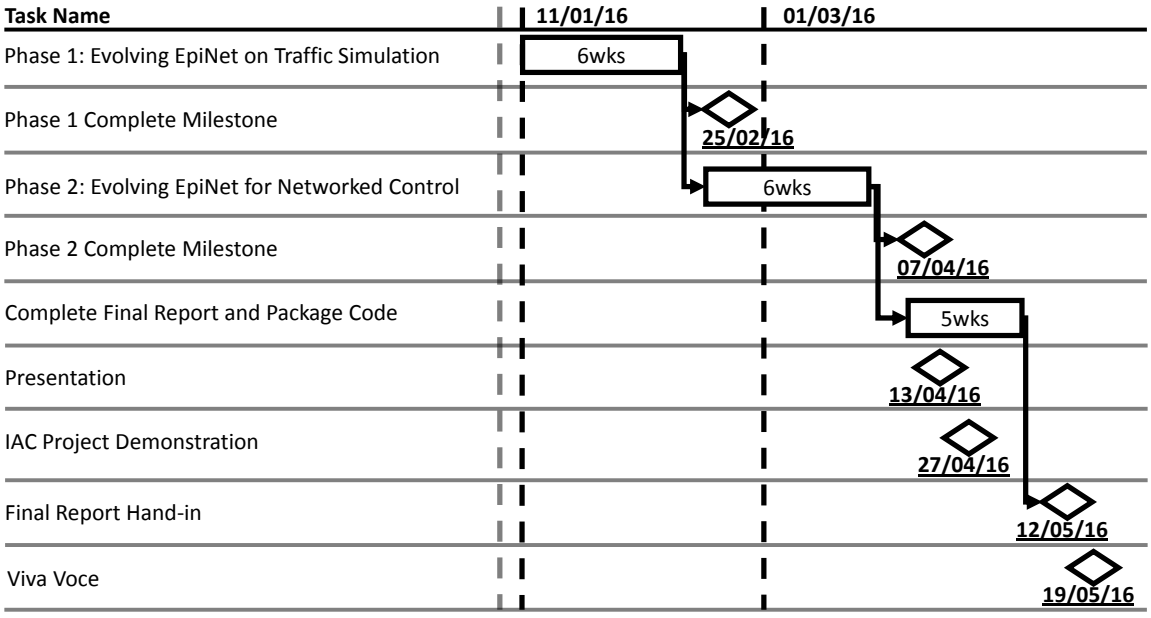

Figure 9.3: Gantt chart for the **final** project plan. High-level tasks are listed at the left hand side. Blocks are used to indicate when each should being, and how long work is expected to take. Arrows show precedence between tasks (the pointed to task cannot start till the predecessor is complete. Diamonds show milestones for the project with their date of completion noted directly below them.

Risk

This project should be considered low risk owing to its scalable nature. A risk register is presented in table 9.1 with; probability (P), impact (I) and risk factor (R=PI). Both P and I

exist on a scale from 1 to 5 where 1 is considered to be almost negligible and 5 is considered significant.

Table 9.1: Risk Register.

Description	**P**	**I**	**R**	**Mitigation**
Impractical computational cost	3	3	9	1) Determine resources early. 2) Develop efficient simulations. 3) Adjust scope.
Excessive difficulty	2	4	8	1) Scalable project. 2) Request advice or assistance.
Poor software quality	3	2	6	1) Plan for testing. 2) Request scheduled inspections.
Missed deadline	1	5	5	1) Short-term goals 2) Scalable project.
Deleted work	1	5	5	1) Regular scheduled backups. 2) Maintain code repository.
Scope creep	2	2	4	1) Well defined requirements. 2) Incremental development.

Three risk points were encountered during development. The first was that implementation of a formal co-evolution methodology required impractical computational resources. There was a failure to anticipate this scenario when planning to use this process in the initial project planning phase. The scope of the project therefore had to be changed such that co-evolution was not investigated, but rather the evolution of separate controllers. Secondly, epiNet was ported to C++ which could have led to a poor, or inaccurate, final program. To mitigate this the software was tested with its original author present such that any deviations from previous tests could be found. This led to some inaccuracies being altered validating the need for such a step. Finally, the aspirational goals of a multi-objective GA and implementation onto professional simulators had to be dropped such that the project met its deadline. This was successfully anticipated in initial planning (hence the goals were aspirational) and therefore re-scoping the project remained simple.

Summary

Project management was successfully conducted on this project and was found to be incredibly helpful. It forced work to become modular with the creation of weekly goals (which were easy to work towards and avoid scope creep). This allowed development times to be predicted and

tracked weekly. When the project began to run behind, or problems were encountered, it allowed for review and mitigating action to be taken. Ultimately this enabled the project to be confidently and successfully completed.

Work Breakdown Structure

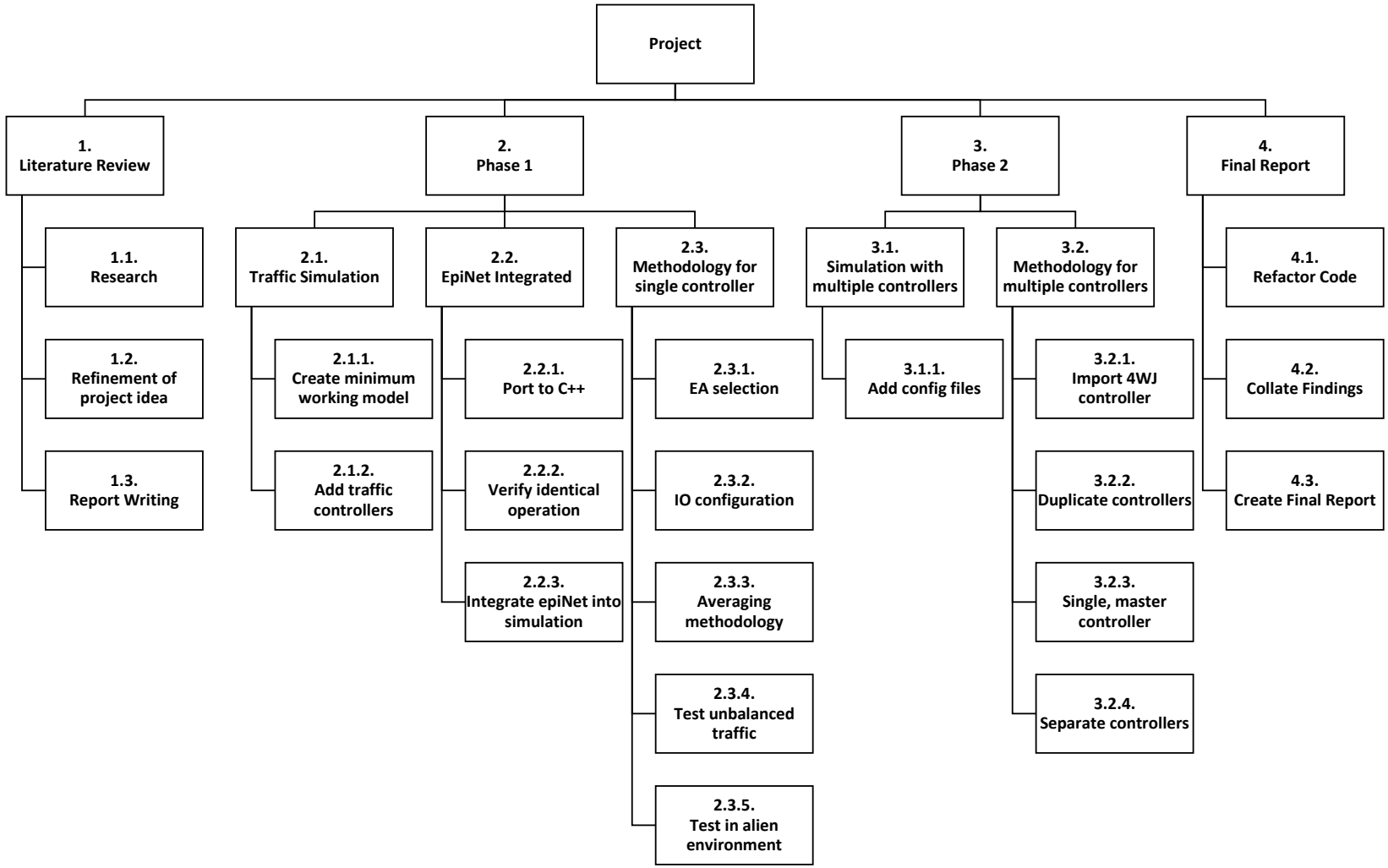

Figure A.1: Final work breakdown structure for the project. Shows the high-level tasks (grouped) that were completed.

Gantt Chart Screenshot

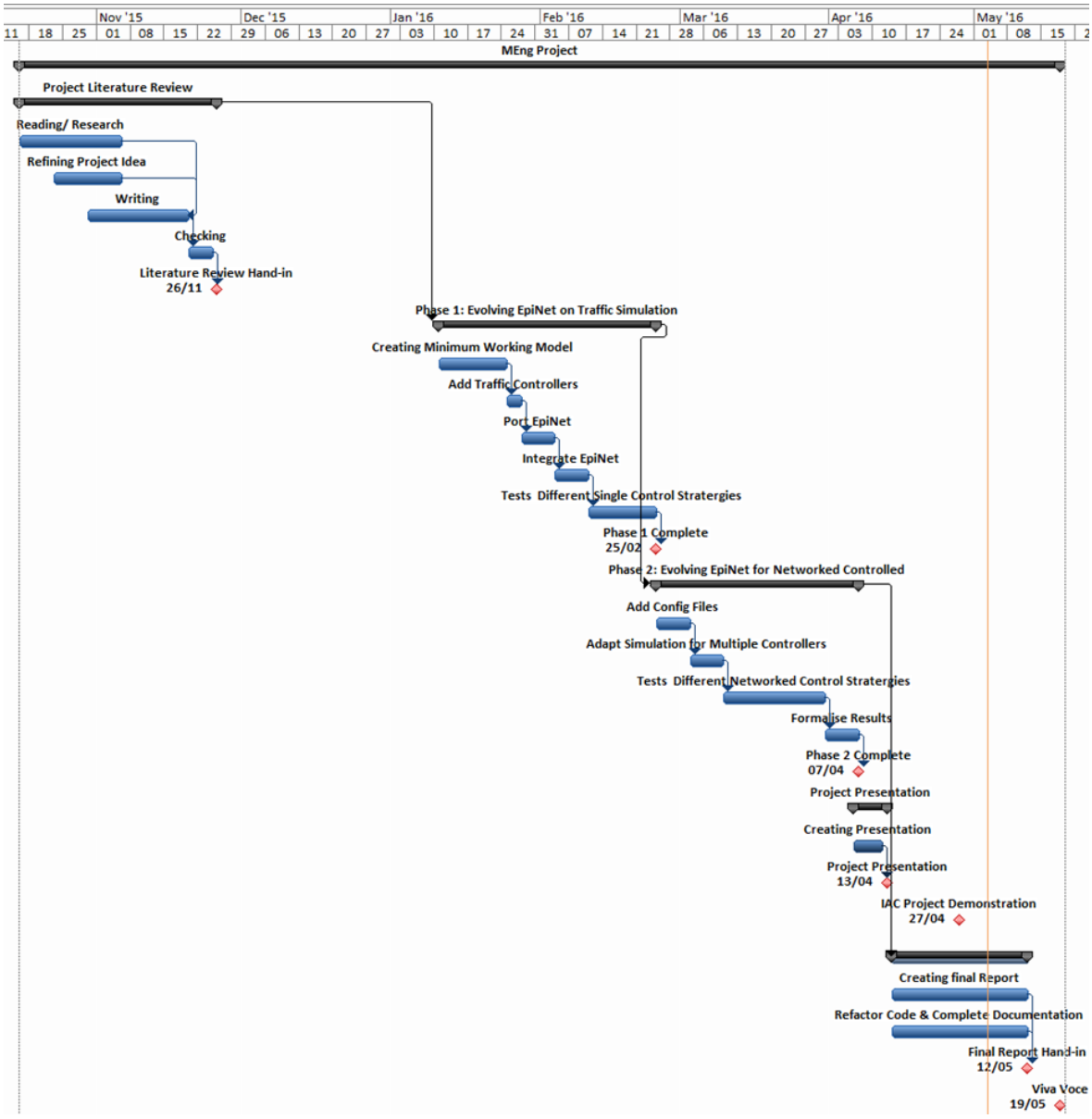

Figure A.2: Gantt chart showing final project time line. *Gray bars* show task groups, *blue bars* show tasks and *red diamonds* show milestones. Each is labelled above, with milestone dates placed at their left. Arrows show precedence.

DLL Export Wrapper Example

Listing 1: C code for the DLL export wrapper

```
#include "C:\Users\jb1296\Source\Repos\simulation\Simulation.h"
#include "C:\Users\jb1296\Source\Repos\simulation\Simulation.cpp"

// Function to CREATE new Simulation
extern "C" __declspec(dllexport) Simulation* newSimulation(void)
{
    return new Simulation();
}

// Funcion to RELEASE Simulation.
extern "C" __declspec(dllexport) void releaseSimulation(Simulation* s)
{
    delete s;
}
```