Distributed Autonomous Intersection Management with Neuro-Evolution

Matthew P. Cherry

July 2021

A dissertation submitted in satisfaction of the requirements for the degree
Master of Science in Computer Science
University of Cape Town

Supervised by: Geoff Nitschke

I know the meaning of plagiarism and declare that all of the work in this document, save for that which is properly acknowledged, is my own.

Abstract

The sudden surge in computational power available to computer research in industry and academia has led to developments in AI automation. More and more tasks are able to be automated and replaced with machine learning systems. One such task that promises to be highly beneficial is that of driving, clearly indicated by the amount of resources being spent by companies such as Uber, Google and Tesla. Neuro-Evolution has shown promise in the field of controller development, due to its ability to develop complex behaviour without a need for any labelled training data. It has been applied previously in car controller generation, across many fields. This thesis aims to apply Neuro-Evolution specifically to the field of intersection management, in order to study which methods are the most effective for this particular task. In particular we investigate three key hyper-parameters: Neuro-Evolution algorithm, task difficulty and problem exposure. A traffic simulator was developed and the hyper-parameters were used to evolve car controllers, which where then tested on unseen tasks. We show that certain key combinations of hyper-parameters yield exceptional results, but that direct correlations between individual parameters and performance are unclear, indicating that these methods are highly sensitive to hyper-parameter selection. We further identify some areas in which to optimize the evolution method, by looking at hyper-parameters which have a computational cost but which did not produce better performance.

Contents

List of Figures

List of Tables

1 Introduction

Self driving cars are currently in advanced stages of development. Companies like Google, Uber, Tesla and Chrysler are aiming to develop a mass consumer market that will eventually replace human-driven vehicles. These autonomously driving vehicles aim to be safer and more efficient than human drivers, making transport cheaper and relieving traffic congestion. This has the potential for numerous benefits not only to humans' quality of life, but also for the efforts of preventing climate change. While simply replacing human drivers with autonomous drivers already offers many advantages, there are even more advantages to be gained by re-imagining the entire transport network, which currently relies on a rules-based system designed to accommodate these human drivers. These rules must take into account human capacity for co-ordination with other drivers, as well as reliance on visual cues such as road signs and traffic robots. Due to limited human reaction times, calculation power and predictive power, these systems are simple and leave room for optimization. Computers however, are able to transmit, receive and process data from many more sources in a matter of milliseconds. This should allow a much more complicated and efficient traffic co-ordination system than is possible with human resources. Through the use of heuristic algorithms and machine learning, computers have already proven the potential for much higher co-ordination ability than humans (Dresner and Stone, 2008; Parker and Nitschke, 2017; Bazzan, 2005). Where humans require traffic lights that only allow two lanes to move at a time, centralized computer controllers such as Autonomous Intersection Management (AIM) have demonstrated traffic that can flow in any lane at the same time (Dresner and Stone, 2008). Investigating further methods for autonomous intersections could provide benefits in other areas. In particular, machine learning could be used as a way to develop a more generalized and potentially higher performing method than those created through heuristics, due to its ability to explore its problem space automatically, finding solutions that may not be intuitive.

The aim of this thesis is to investigate one such possible algorithm, namely Neuro-Evolution (NE). This solution involves evolving a neural-network driven controller that, through the use of various sensors, can control the vehicles in an intersection and create emergent co-operative behaviour. NE would allow a solution which is distributed between cars with no central authority and which could be generalized to work across most intersections, including those that it has not been specifically designed for. NE is a complicated algorithm with many parameters and human decisions to be made during implementation. This research attempts to experiment with three different NE hyper-parameters including the specific NE algorithm used, task difficulty and problem exposure, in order to help understand further the effectiveness of NE in this space.

1.1 Motivation

Human drivers are prone to mistakes, which cause accidents on the roads. Furthermore, humans have to rely on a limited set of visual and audio cues, through a limited set of sensors, for co-ordination with other drivers. This has adverse effects on traffic, such as when bad communication and sudden rash movements cause traffic where there is none, or where an intersection becomes gridlocked, causing traffic jams. Computers specialized in driving therefore could allow these issues to be solved. Through wireless networks as well as individual processing, computers are able to much more effectively co-ordinate and make decisions about how to act on the road. The exact architecture of this computer system however must be chosen carefully. A central controller managing an intersection would provide a single point of failure that is susceptible to attacks and failures. A heuristic controller would need to be tuned for particular intersections and may fail to take into account scenarios that the designer did not consider. NE is a promising answer to these issues due to several key characteristics.

NE (Floreano et al., 2008; Stanley and Miikkulainen, 2002; Stanley, D'Ambrosio, et al., 2009) has been shown to be effective in evolving controllers for co-operative agents, both in computer simulations (Agapitos et al., 2007) and physical robot systems (Salichon and Tumer, 2010), and has already been applied specifically, with some success, to intersection management (Parker and Nitschke, 2017). This suggests that NE is a viable way for developing solutions to autonomous intersection management. A NE solution would be beneficial due to its distributed nature, which would help proof against tampering and failure, as well as its generalizability. Each vehicle has its own controller operating at any given time, meaning that in most cases a failure could only result in one vehicle producing abnormal behaviour. There is also no single entry point for malicious attacks, besides more elaborate attacks that involve the distribution of controller code. NE controllers are also not hand crafted, but evolved. This evolution process can be designed in such a way that controllers must develop generalized behaviour which can adapt to many situations, even those that were not considered by the designer of the controller.

As beneficial as these features may be, they come at the cost of complex machine learning required in developing the controllers. Unlike a heuristic algorithm, NE relies on search algorithms to find candidate solutions. These algorithms are stochastic, can consume a large amount of resources, and have a large number of parameters that must be tweaked to develop optimal solutions. Specifically in NE, these search algorithms make use of evolutionary principles. To understand further how best to use NE for the problem of autonomous vehicles, evolution parameters must be tested. By exploring and analysing these parameters, results can be obtained which will help in the future use of NE in the problem of autonomous vehicles. This can range from the performance or viability of different NE algorithms, to specific evolution parameters to be used to obtain useful controllers.

This thesis aims to provide this analysis, and help future researchers understand how to use NE algorithms to effectively develop intersection management controllers. The hyper-parameters analyzed in this thesis, namely: algorithm, problem difficulty and problem exposure, all can directly affect not only the performance of the autonomous system during evolution but also their generalizability, and are key factors during development. By understanding which algorithms perform better, and which parameters produce better controllers, less time can be spent on evolution and development.

1.2 Research Goals

The primary goal of this thesis is to compare sets of hyper-parameters in NE evolution to determine which ones lead to the best task performance, and to understand how these hyper-parameters affect task performance. Hyper-parameters here include the NE method itself as well as evolution environments. Task performance is measured by a solution's ability to allow throughput in a traffic system with as few collisions as possible.

To this end, several key hyper-parameters in evolution were chosen as independent variables for comparison. These hyper-parameters are algorithm, problem difficulty, and problem exposure. There are many other hyper-parameters not tested, due to limitations on computer power, however results from this study will still help to inform which of these hyper-parameters to explore.

For each hyper-parameter, three possible values were compared. This lead to a total of 27 different evolution regimes being used to develop controllers for a homogeneous set of cars interacting in various intersections. The regimes consisted of every permutation of the three hyper-parameters, as seen in Chapter 4.1 and Table 3. They are:

- Algorithm - CNE, NEAT, HyperNEAT
- Problem Difficulty - Easy, Medium, Hard
- Problem Exposure - One Intersection, Two Int., Three Int.

Three different NE algorithms have been selected to explore the Algorithm hyper-parameter. They are Conventional Neuro-Evolution (CNE) (Whiteley, 1988; Montana and Davis, 1989; Whitley et al., 1993), Neuro-Evolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002) and HyperNEAT (Stanley, D'Ambrosio, et al., 2009). These algorithms are described in more detail in Chapter 2. This hyper-parameter is most likely to generate very large differences due to the way it affects the search process, and understanding it is crucial in any further work in the field.

Problem difficulty is an important hyper-parameter in determining the nature of scenarios during evolution. It also has important implications for consumption of computer resources, as it multiplies the amount of physics that must be simulated in every single fitness evaluation. By understanding what problem difficulty to evolve with, much more efficient evolution could be performed in future.

Problem exposure is not only important in computer resource consumption but also in understanding controller behaviour. This hyper-parameter gives insight into how well NE can generalize and how much evolution is required to develop controllers capable of being practically useful in as many scenarios as possible. Problem exposure directly multiplies the amount of trials that must be run to evaluate fitness, thus there is a linear correlation with computer resource consumption and problem exposure.

The crux of this research is to analyse the performance of the controllers developed using these differing hyper-parameters, and investigate the impact of the three different evolution factors. The hyper-parameters were measured by their performance on unseen tasks, their controller complexity, their behavior complexity and their exploration of the problem space during evolution. Some analysis of evolution performance is also given. These results should provide insights regarding the evolution of controllers for intersection management, which hyper-parameters are important, and how to select hyper-parameters for similar tasks.

1.3 Methods

In order to test and compare the three hyper-parameters, a set of experiments was conducted in which NE was applied to each permutation of the hyper-parameters, in order to develop car controllers. Thus, each combination of possible hyper-parameters, namely algorithm, problem difficulty and problem exposure, will be used in an individual evolutionary development task. These experiments are detailed in Chapter 4. To develop and analyse these NE controllers, an analogous representation of an intersection with traffic flowing through it was needed. This representation was used as the context in which to evolve controllers, and then test them. A detailed description of the simulator is provided in Chapter 3. After evolving the controllers, they were subjected to a set of unseen tasks in order to gather data about their performance in several respects. The controllers were also compared alongside a baseline heuristic controller.

Several sets of data were collected and analyzed from both the evolution and testing phase of the experiment. Generational fitness was collected during evolution, and unseen task performance, behaviour statistics and network complexity were collected during testing, as presented in Chapter 5. Where needed, statistical significance was tested for.

1.4 Contributions

This research aims to determine the optimal hyper-parameters to be used in developing autonomous intersection management controllers through NE, and to understand how these hyper-parameters affect evolution and controller performance. Performance and sensitivity analysis of these hyper-parameters will lead to an understanding of how each contributes to final controller performance. Since most of these hyper-parameters have resource usage implications, it is useful to know if hyper-parameters do not affect performance so that the cheapest value can be used.

These results will be able to be used in future experiments in order to develop even more advanced controllers. The research will also shed light on the viability of NE as a solution to autonomous vehicles. By drawing on these results, future research may be able to fix many of the variable factors in experiments, reducing the number of permutations needing to be tested, and therefore reducing the amount of time and money spent.

1.5 Structure

First, in Chapter 2, we go through the background research of the field of NE, in order to describe and understand the various NE algorithms we applied, as well as comparative research in the area of intersection management. The simulation method mentioned above is described in depth in Chapter 3, and was used to execute experiments in order to gather data for the research goals. Chapter 4 describes the exact nature of these experiments and how they were executed. Chapter 5 then deals with collecting results from experiments, processing them, and presenting their outcomes. Chapter 6 discusses these results and relates them back to the initial research goals. Finally we conclude with key takeaways from experiments, as well as potential future work in Chapter 7. The thesis is followed by acknowledgements, references and several appendices.

1.6 Conclusion

Intersection management presents an intriguing, relevant task with which we can explore the methods of NE. In order to explore these methods and collect data, a simulator has been built, and various experimental configurations have been executed. We now go through the relevant background research before moving on to describe the methods and experiments used in more detail.

2 Background

This thesis is in essence an exploration of the application of several NE algorithms to controller development for multi agent systems, in the context of intersection automation. Hence, this background section aims to layout the key terms and techniques used in NE as well as previous research and progress in the area of multi agent systems, machine learning controller development and traffic automation.

2.1 Traffic Automation

Modern conventional traffic systems rely on a set of easily identifiable signals that can be seen and interpreted by human beings driving vehicles. This, combined with a set of traffic rules, enables large amounts of people to co-ordinate effectively in their individual tasks of travelling, be it through dense city centers, quiet suburbs or out in the countryside on the highway. The value in having a general traffic system open to any member of the public with a valid licence cannot be overstated, however it brings with it some challenges.

The World Health Organization's Global Status Report on Road Safety in 2018 showed road deaths as the highest cause of death of people aged 5 to 29 years old. The INRIX 2019 Global Traffic Scorecard showed that congestion costed the UK economy £6.9 billion for that year. Clearly congestion and road safety are two key areas in which improvements to the traffic system could be made, which would provide huge benefits to society.

In this section, the application of computer algorithms to improve traffic systems is explored. By understanding these systems we can form a base from which to envision a future traffic system which may not require conventional traffic signals at all.

2.1.1 Steering and Lane Management

A seemingly low hanging fruit in the process for developing full traffic autonomy would appear to be that of steering the car in controlled circumstances. Such circumstances would include a car travelling along an open highway or following a small road. In these cases, no complex decision making is required, and co-ordination between drivers is minimal. Yet the benefits of automating this area of traffic are huge. Human drivers are prone to errors caused by tiredness, intoxication, distraction and bad judgement, any of which could prove fatal on the open road. Having a computer system manage steering and road lane changes could potentially solve these issues.

Due to the relative simplicity of the problem, and the discussed potential benefits, this area has been well researched, with commercial options already available to the public. Early research began by developing simulated and real world systems that modeled vehicles on the road and implemented straight forward algorithms for path following and lane switching. Overall there appear to be two general approaches to this, one using machine learning algorithms and the other using hand made algorithms based on modeling and heuristics. In terms of heuristic systems, which allowed lane following and switching, they resemble more traditional engineering control solutions, and were successfully installed in actual cars, as shown in Figure 1 (Reynolds, 1999; Hatipolglu et al., 1997). Flow dynamics were also used to model potential automatic highway systems (Alvarez et al., 1997).

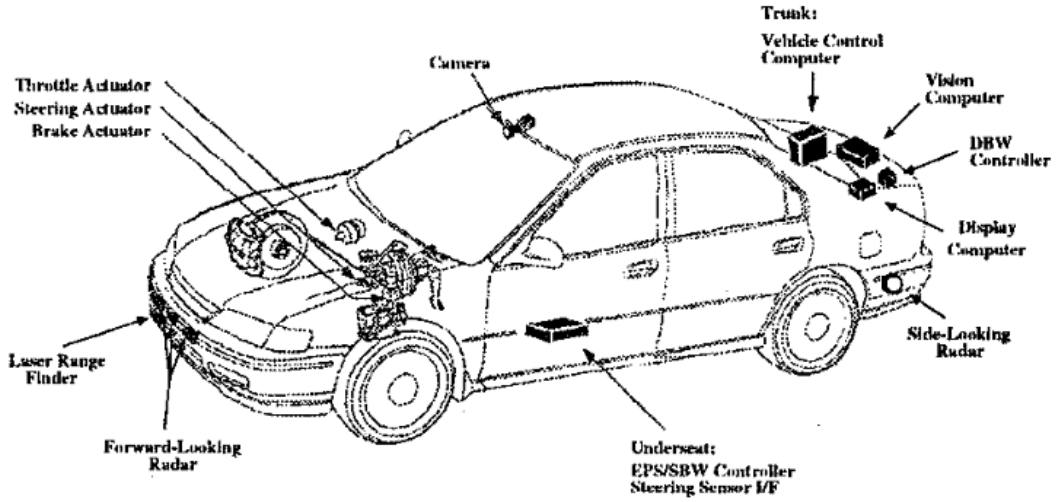

Figure 1: A traditional approach to autonomous vehicles. Sensors and outputs are connected through a controller which is programmed by engineers. The vehicle is outfitted with the sensors and control system (Hatipolglu et al., 1997).

On the other hand is the approach of using machine learning. The system developed by Pomerleau (1992), makes use of trained Artificial Neural Networks (ANNs) to develop a controller for a van that is able to follow a road based on camera data. Furthermore, to optimize the flow of traffic on highways, Moriarty and Langley (1998) used NE to develop lane selection systems that could optimize throughput.

Through multiple approaches the problems of lane keeping, lane switching and even lane selection have shown to be optimizable by a combination of heuristic and machine learning algorithms, as early as 1997. It is no surprise then that this research has culminated in technology such as Tesla's Autopilot, a commercially available system that allows Tesla vehicles to follow roads, switch lanes, and even avoid obstacles. This first step towards automated traffic is a large one, and sets the precedent for further automation of the traffic system. While Tesla Autopilot is a working reality, it is still unable to make use of all the works mentioned above, do to its requirement to operate in a still-human system. Once this is overcome, systems such as those developed by Moriarty may become possible, as well even further optimization in the other areas of traffic management, namely intersections.

2.1.2 Traffic Signal Optimization

Conventional traffic systems operating at intersections make use of traffic signals, most often given by traffic lights and signs. These systems are designed to be usable by humans and rely on a very simple set of rules. It stands to reason that a computer system could operate with a more complex and robust set of rules, or even potentially co-ordinate directly, bypassing the need for traffic signals at all. While completely removing the need for traffic signals is appealing, advancements in the effectiveness of current systems should still be considered first as a bridge towards fully automated intersections.

Several well known systems exist and are in use with current traffic systems such as SCOOT. These systems, while based on traffic data are flawed in their ability to react to sudden differences in traffic in real-time (Dresner and Stone, 2008; Roozemond, 1999). There are clear improvements to be made with real-time processing computer systems that are able to work with data available in the traffic system. One such system is proposed by Roozemond (1999), in which intersections are modeled as autonomous agents that are able to process and share data between themselves. The key benefit being the ability to make use of real-time traffic network data. This line of thinking is further extended by Bazzan (2005), in which autonomous intersection agents are modeled with game theory. In Bazzan's proposed systems, agents aim to optimize local and global throughput of the traffic network simultaneously. A somewhat different approach is a system making use of reinforcement learning systems, proposed by Abdulhai et al. (2003). In this research, instead of heuristics and modeling, optimal traffic signals are learned through Q-learning, a simple type of reinforcement learning. Such a system could in theory learn continuously and adapt to changing traffic conditions, as the development of the traffic signal controller is semi-supervised.

More recently, cellular automata and traffic modeling solutions have shown to be highly effectual at improving traffic flow in large systems (Gershenson and Rosenblueth, 2012). They use a simplistic model where traffic is modeled as boolean values on a one dimensional plane, and time steps discreetly. The value of each boolean node is a function of the previous time step and neighboring nodes. This allows a set of simple rules to develop emergent and complex behaviour which are shown to be effective at organizing traffic. These systems have been applied to simulators making use of real world data, with results showing that wait time in the intersection can be reduced by a half (Cools et al., 2013). These systems are hand crafted models which make use of heuristics. As such they are limited by human ingenuity.

A final note on this topic, is the replacement of a need for humans to interpret and react to traffic signals. Such a system would be pivotal in transitioning between traffic signal use, with mixed human-artificial driver traffic, and fully autonomous traffic. Research by Lindner et al. (2004), showed the application of computer vision algorithms in deciphering and tracking traffic signals. This system is depicted in Figure 2. Such a system, combined with automated lane keeping, makes it possible for most traffic operations to be performed by a computer driver operating in the current human-centric traffic system. While this provides many benefits, and removes human error, there is a great deal of optimization possible with further steps towards completely autonomous intersections that do not need to take into account human limitations at all.

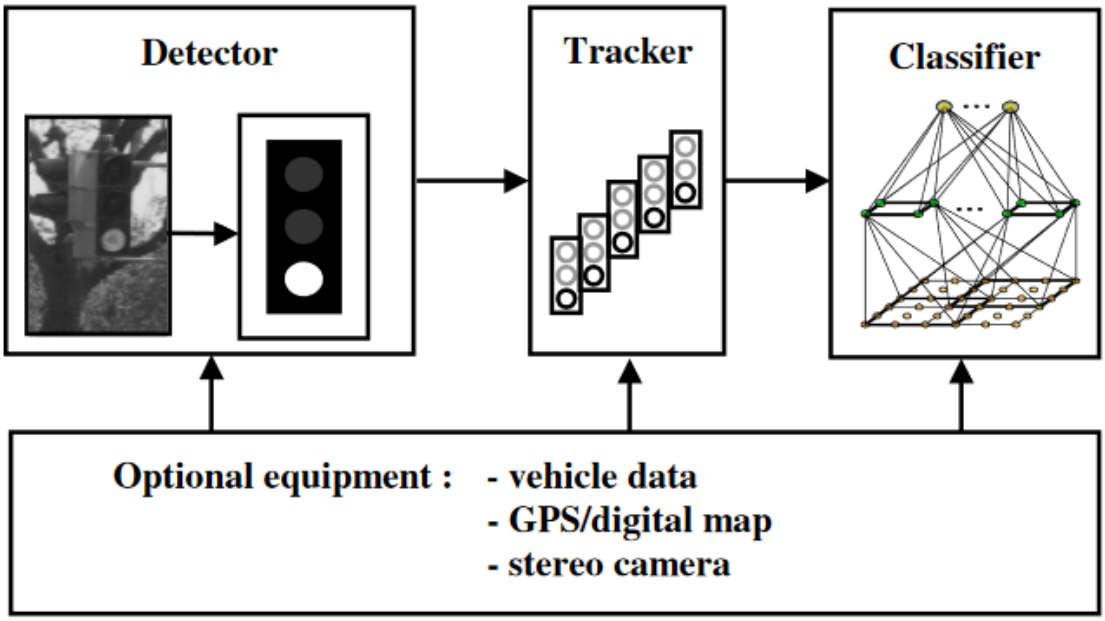

Figure 2: Computer vision system architecture for traffic signal detection and recognition (Lindner et al., 2004). Such systems mostly make use of convolutional neural networks to identify common patterns in images.

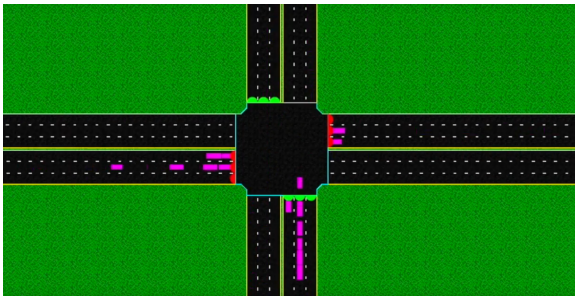

Figure 3: AIM implemented in a simulator (Dresner and Stone, 2008).

2.1.3 Intersection Automation

While there are obvious benefits with removing humans from the equation using the methods above, things can be taken further by leveraging the power and coordination ability that computer systems are capable of. In a traffic network fully operated by computer systems, safety could theoretically be increased without sacrificing throughput. In fact, it is most likely that throughput and efficiency will be increased. These systems would control every decision and interaction that takes place in a traffic intersection, a difficult problem considering the complexity of decision making required as well as limitations on traffic bandwidth. Methods that have have been implemented towards automated intersection management are discussed below.

Several heuristic based intersection systems have been proposed (Naumann et al., 1997; Dresner and Stone, 2004; Dresner and Stone, 2008). An early iteration by Naumann et al. (1997), presents a working decentralized intersection heuristic, based on the principles of semaphores and locking in multi-threading. Vehicles in the intersection attempt to obtain locks that allow them to pass through certain regions of the intersection, with only one vehicle being allowed to have any specific lock at a time. In theory this prevents any possible collisions in the intersection. While the decentralized nature of this algorithm is desirable, it has several flaws as pointed out by Dresner and Stone (2008). Two of which are the susceptibility to communication issues between vehicles, and a lack of efficiency due to a simplistic algorithm that does not plan very far in advance. Dresner instead proposes a centralized heuristic algorithm called Autonomous Intersection Management (AIM), which makes use of a reservation based system in order to manage cars passing through the intersection (Dresner and Stone, 2004; Dresner and Stone, 2008).

This centralized system is robust and allows for careful optimization of cars passing through the intersection. By using a first come first serve reservation system coupled with a simulator capable of working out if any reservation could cause a collision, AIM is able to allow many cars to enter the intersection at the same time, from different roads while guaranteeing no collisions. Furthermore, work presented in 2008 shows direct comparisons between AIM and traditional traffic signal systems, with a substantial improvement shown in the results (Dresner and Stone, 2008).

Finally, a machine learning approach was taken in Parker and Nitschke (2017) with direct comparisons to AIM. The methods in this thesis will be examined in the next section of this background, however there are some key takeaways with regards to performance and implementation. Firstly, results proved to be promising with NE outperforming AIM in several scenarios. Unfortunately separate controllers were used for each intersection, leading to results not taking generalizability into account. At the very least however, NE shows the potential to outperform a heuristic method such as AIM for given intersections.

It is worth noting several other systems in this area proposed that do not solve intersection automation wholly, but rather improve certain aspects of it and which could be combined with other systems. First is Hallé and Chaib-draa (2005), which proposes a set of heuristic systems that make use of platoons - vehicles grouped together and working in unison - to perform certain maneuvers in an efficient way. An example being a backed up set of cars driving through an intersection. Whereas human driven cars will wait for each car in front of them to start moving, platoons would allow all vehicles to begin moving simultaneously, allowing higher throughput.

Kohl et al. (2006) presents a vehicle warning system developed through NE that aims to prevent vehicle collisions in a variety of circumstances during intersection navigation. The proposed system makes use of simulated cameras and develops a controller through evolution using the NEAT algorithm, one which is used in this thesis.

We have seen the potential for full intersection automation, which combined with highway and open road automation, could form a highly efficient and safe traffic network, saving lives and billions of dollars. We have also seen NE as a promising solution to some parts of this automation problem. This thesis attempts to further explore NE as a possible solution for intersection management as a whole, and is discussed in detail further below.

2.2 Neuro-Evolution

In this section we explore the field of NE. NE is a biologically inspired set of techniques which draw from two other biologically inspired fields, namely ANNs and EAs. We shall see that NE provides a powerful technique for developing controllers in an unsupervised manner, which can be used to attempt to solve the problem of intersection automation. First the precursor fields of ANNs and EAs will be explained, followed by NE techniques themselves. Finally, we will examine previous work which has made use of NE in similar tasks to the one in this thesis.

2.2.1 ANNs

ANNs are a mathematical system inspired by biological brains (Svozil et al., 1997; Haykin, 2004). As early as the 1940s, work has been done to try formalize the operation of brains mathematically (McCulloch and Pitts, 1943), but one structure has stood out as being the most popular and widely used. This is the ANN. Key features that make the ANN so widely applicable and widely used are its ability to act as a universal function approximator (Maggiora et al., 1992), its ability to be setup with an arbitrary topology with any number of inputs and outputs, and the ease with which they can be trained. While there are many different forms of ANN, we will be focusing in this thesis on multi-layer feed forward networks, as described in Svozil et al. (1997) and Haykin (2004). Wherever we refer to an ANN, it will be a multi-layer feed forward network specifically.

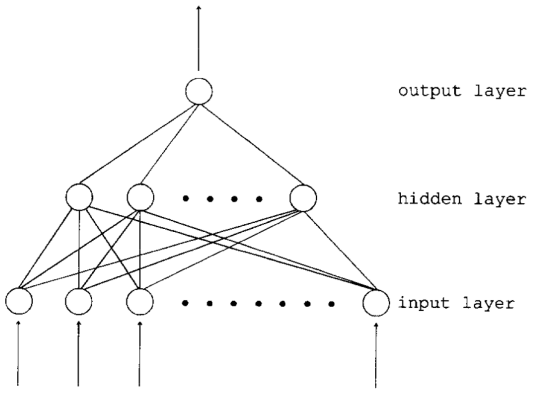

Figure 4: Structure of a multi-layer feed forward network. Each node is a circle, with connections shown as arrows between nodes. Input is fed into the bottom layer and propagates upwards to the output layer. (Svozil et al., 1997)

The base structure in an ANN is a node. A node is able to accept a number of inputs, and produce an output that is a function of the inputs. This function is usually homogeneous and is called the activation function. In a multi-layer feed forward network, these nodes are arranged in ordered layers with two special layers called the input and output layers respectively. The input layer is the first layer in the network, and its node inputs directly represent the input for the entire ANN. The output layer is the last layer, and its node outputs present the ANN's solution for the given inputs. Layers in between the input and output layer are called hidden layers, containing hidden nodes. Between each layer, nodes can be connected by a directed weighted edge, allowing one node's output to feed into another node's input. In a feed-forward network, these connections are strictly in the direction from first layer to last layer. We can therefore consider the ANN to be a directed, weighted graph, where each node is a vertex and each connection is an edge. Figure 4 shows a fully connected ANN as a graph structure. This structure can be seen to naturally represent a robotic controller, capable of ingesting sensor data, passing this to the ANN's input layer, and then calculating the output layer which can be used to control physical processes.

It is also possible for an ANN to be modeled as a series of matrices, which can be calculated with matrix multiplication. We shall use this to formally describe how an ANN may be computed. Given an activation function *f*, a layer l of n nodes (which we will take to be a vector of length n), the next layer in front it *l+1* of m nodes (again, as a vector), and a matrix of $m \cdot n$ weights w (where $w_{j \cdot k}$ is the weight between the j th node in l and the k th node in *l+1*), in a fully connected ANN, we can calculate the output of *l+1* as follows:

$$l + 1 = f(l \times \begin{bmatrix} w_{1 \cdot 1} & w_{1 \cdot 2} & ... & w_{1 \cdot m} \\ w_{2 \cdot 1} & ... & ... & ... \\ ... & ... & ... & ... \\ w_{n \cdot 1} & ... & ... & w_{n \cdot m} \end{bmatrix})$$

When we calculate an ANN, the input layer must first be set to match the problem state. From there, we can iteratively use the above calculation to calculate each layer starting from the first hidden layer all the way until the output layer. Also note, that in the case of a non-fully connected topology, we can consider a weight to be 0 where there is no connection between nodes.

While the above methodology enables a universal function approximator, there are two glaring issues to deal with. The first is the exact topology of the ANN, namely how many hidden layers to use, how many nodes to have in each layer, and which activation function to use. The second is how to determine the correct weights to use. As we have seen, this is a key component in determining the outputs of an ANN. In terms of topology, this can be either manually chosen, or developed as part of the training regime. We shall cover these techniques in in more depth in the NE section of this background.

Determining weights on the other hand, cannot be done manually, and so a sophisticated mechanism is needed to allow these weights to be determined without direct human intervention. One such system, widely used in data science for predictive and classification tasks, is backpropagation through errors, first presented by Rumelhart et al. (1986). Backpropagation through errors trains the weights in a network by feeding in input vectors alongside output vectors, also known as labelled training data, which represent a correct solution for the given input. Using differential calculus, the difference between the actual ANN output and the correct output vector (called the error) is propagated backwards through the network and used to shift the weights. This technique is effectively a form of gradient descent, and can be highly effective for large input-output data sets. One large drawback however is the requirement of correct output vectors, which must often be curated manually. This drawback makes back propagation through errors a difficult technique to apply when developing controllers, as an optimal output vector is not obvious to determine for a given set of inputs. Instead, an appealing algorithm for ANN-based controller development would be one requiring no labelled training data. Such an algorithm is possible, and discussed further in the next section.

2.2.2 Evolutionary Algorithms

Where ANNs take inspiration from biological brains, Evolutionary Algorithms (EAs) take inspiration from the biological processes that evolved the brain. An EA uses the concepts of survival of the fittest, adaptation, and genetic mutations and re-combinations and applies it to the concept of search spaces. The key characteristics of an EA are its non-optimal stochastic nature, wide applicability, and lack of requirement for any labelled training data.

EAs have been studied for decades, with the first experiments being conducted by Holland (1975). Since then the concept of adaptive computational systems has led to what we understand as a Genetic Algorithm, as described by Whitley (1994). At its core, a genetic algorithm has an iterative cycle, in which a pool of candidate solutions are evaluated, selected and finally modified and recombined to form a new pool. Initially the candidate pool is generated randomly. Each candidate solution is encoded in some form, canonically a fixed-length binary string, which can be used to evaluate the fitness of the solution. This fitness evaluation is a key function, and must be selected carefully to incentivize more optimal solutions. Once candidates have been evaluated, the ones with the highest fitness are selected to form a new candidate pool, called a generation. There are many potential selection algorithms, but generally they aim to probabalistically choose higher fitness individuals. In some cases, completely new randomly generated candidates are added to the new candidate pool. After selection, all candidates solutions are subject to mutation and crossover functions, with some probability. Mutation functions alter a candidate solutions encoding directly by some function, for example swapping two values in the encoding. Crossover functions attempt to merge two candidates' encodings by selecting certain sections of the encoding and swapping them with the other candidate. In theory, the candidate pool should become fitter as the GA cycle is repeatedly applied, and will eventually terminate either at a preset fitness or after a preset number of iterations. Figure 5 shows this cycle.

The power in this algorithm is the ability to explore complicated high dimensional search spaces with only a fitness function. No data or known answers are required, and no part of the evaluation itself needs direct integration with the algorithm. Any problem which can have a fitness measure and can be encoded is able, in theory, to be optimized by the use of an EA.

The difficulty in applying GAs comes in selecting productive hyper-parameters, any of which can greatly affect the final fitness. These hyper-parameters include the size of population to use for each iteration, how many iterations to execute, what probability to mutate with, and so on. Also of key importance is selecting a fitness function that correlates well with task performance, which is not prone to exploitation, as well as designing an encoding that allows desirable candidate solutions and provides a continuous multi-dimensional search space to be explored. Encoding schemes have led to derivatives of the canonical GA such as Genetic Programming, as presented by Willis et al. (1997). In Genetic Programming, instead of a fixed length encoding being used, a tree structure is used which encodes a logic that can be used for controllers.

In this thesis, we use EAs that directly make use of the canonical GA, as well as more specialized algorithms that more closely resemble Genetic Programming, or even further indirection. Our specific end goal for all these algorithms, is to provide a way to develop an ANN, without the requirement for data. EAs that output ANNs collectively make up the field of NE, which we will now give background on.

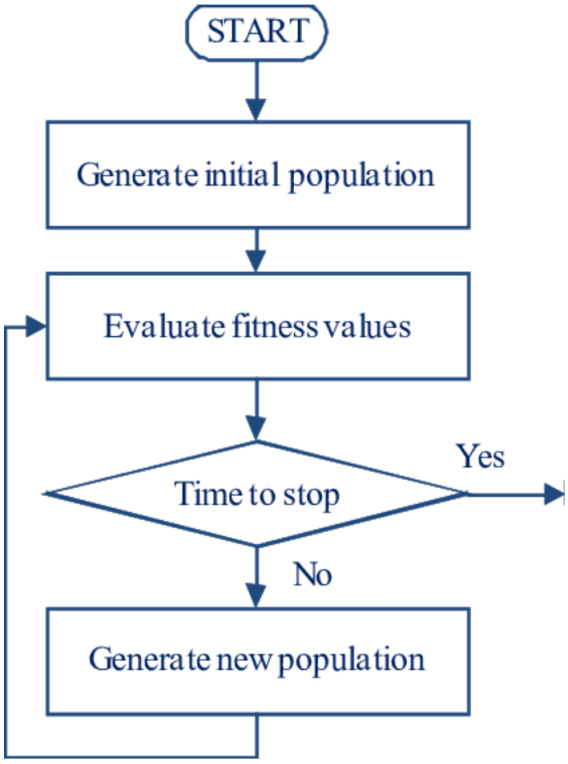

Figure 5: The core steps in an Evolutionary Algorithm (EA). Key development and problem solving is created during the cycle between evaluating fitness and creating new generations.

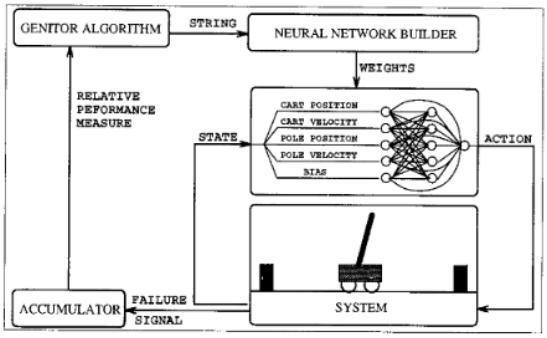

Figure 6: Genetic reinforcement experiment by Whitley et al. (1993). An ANN is developed by feeding signals from a pendulum experiment to an accumulator which reinforces desirable behaviour.

2.2.3 Neuro-Evolution Techniques

The first NE techniques developed made use of canonical GAs and simply applied them to an ANN topology (Whiteley, 1988; Montana and Davis, 1989; Whitley et al., 1993). In this scheme, the GA encoding is a list of weights in a fixed topology ANN. Each ANN is evaluated by mapping the encoding to specific weights in the ANN, and evaluating the ANN's task performance to determine fitness. We refer to such NE implementations as Conventional Neuro-Evolution (CNE) throughout this thesis. The first reported attempted application of GAs to ANN development was by Whiteley (1988), which was unsuccessful. Soon after, Montana and Davis (1989) was able to successfully apply the algorithm to a sonar image classification problem, with results at the time outperforming backpropagation methods. Their method made use of a canonical GA, where each individual was encoded as a fixed-length string of weights. They then made use of a series of standard GA operations such as mutation and crossover, as well as gradient based functions.

While the above mentioned thesis successfully applied the first NE algorithm, it was to a supervised learning task. Whitley et al. (1993) took NE a step further into the domain of reinforcement learning. Reinforcement tasks generally do not have labeled data sets for training. In this case, the task was a simulated inverted pendulum controller. Again, a canonical GA was used with a fixed-length encoding of weights that encoded a fixed-topology ANN directly. The key difference between the previous thesis was the use of a fitness function based on performance during a randomized simulation, instead of error measurements from known solutions. Most controller development tasks are modeled well as reinforcement tasks instead of supervised learning tasks, and so this work presented promising results for the application of NE directly to tasks such as decentralized traffic automation. Whitley's genetic reinforcement method is show in Figure 6.

CNE methods have been developed further, such as in Gomez and Miikkulainen (1997), however fundamental restrictions to GAs prompted development of more advanced algorithms. A key limiting factor of GAs is their restriction to fixed-topology networks and this led to the development of NE algorithms which evolve more than just the weights of the network.

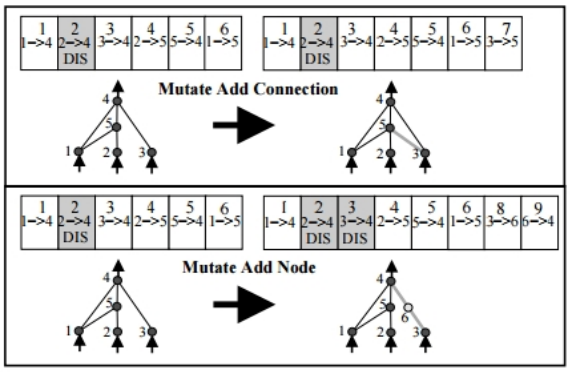

Figure 7: Neuro-Evolution of Augmenting Topologies (NEAT) mutations allowing changes to the topology of the network (Stanley and Miikkulainen, 2002)

One of these additional elements to evolve is network topology itself. The theoretical benefit of evolving topology as well as weights, is to firstly allow greater flexibility in topology. A fixed topology can only have as many nodes or layers as is decided upon. Secondly, evolving topology should lead to a more efficient evolution process. Smaller networks are faster to compute, and by using a strategy which starts with a simple topology, ANN evaluation should remain as cheap as possible throughout evolution. In practice, this means that the trade off of network complexity versus computational cost can be determined automatically by evolution, instead of being fixed prior to evolution. One such system, proposed by Stanley and Miikkulainen (2002), attempts to achieve exactly this. This algorithm is called Neuro-Evolution of Augmenting Topologies (NEAT) and improves upon a traditional GA by introducing several key new features. Firstly, some basic new mutations are introduced, which allow direct manipulation of the network topology. These are namely *add node* and *add connection*, which are self explanatory and shown in Figure 7.

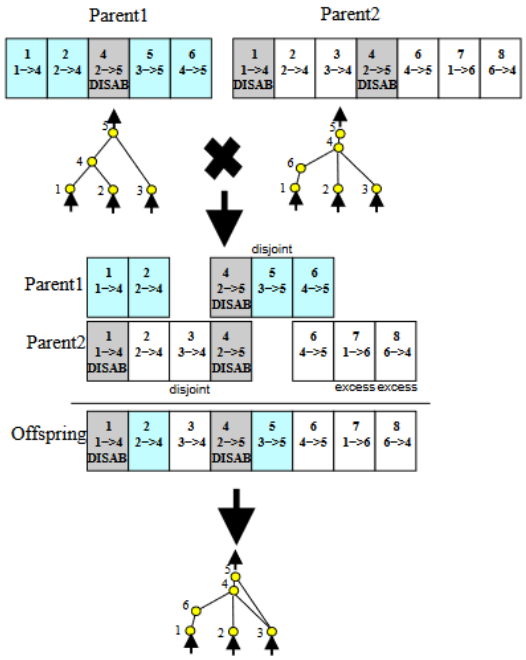

Figure 8: Innovation numbers can be used to match up parts of the topology of two different parents and produce offspring that maintains the original functions of both parents (Stanley and Miikkulainen, 2002)

Also included in the encoding however, is an innovation number which is determined globally for the entire evolution process. As new connections and nodes are added, they are assigned an innovation number, which can then be used to determine which genes historically match each other, as show in Figure 8.

Secondly, a mechanism for protecting the structural development of the ANN is introduced called *speciation*. The differences between individuals during evolution, as a function of their differing weights and innovation numbers, allows grouping into separate species. These species then compete separately for selection. This protects new innovation in the population which may at first be less fit, but which potentially could develop new solutions.

The advanced NE systems in NEAT remove the limitations of fixed-topology networks, however some aspects of the network are still fixed. Input and output nodes must still be preset during evolution, and if new sensors or outputs are wanted, the evolution must be redone. Furthermore, if a complex internal hidden network is required, NEAT's strategy of starting with a minimal network may prove to be inefficient, as it has to slowly develop more complex topology.

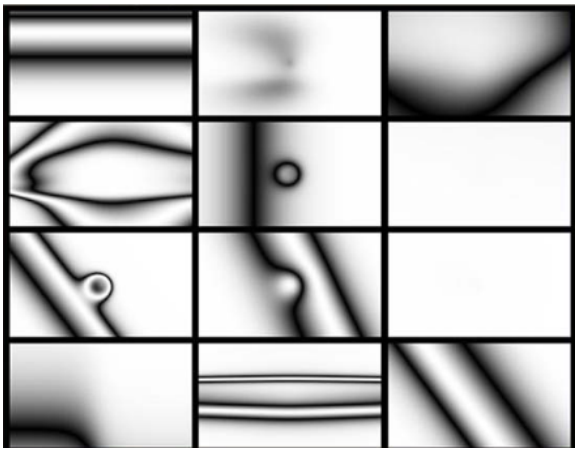

Figure 9: Output, as a function of the second positional parameters, of the Compositional Pattern Producing Networks (CPPN) where the first positional parameters are fixed (Stanley, 2007). Since the CPPN in this case is two dimensional, its output can be plotted as an image, providing a visual representation of the CPPN output.

Both CNE methods and NEAT discussed above make use of an encoding which directly relates to an ANN. As mentioned, this creates a drawback of inflexibility with regards to input and output structure. Because the encoding is direct, any changes require a new evolutionary procedure. In real life applications such as robotics, it is possible for sensor configurations to change however. To address this, Stanley, D'Ambrosio, et al. (2009) proposed a new form of encoding which can be used, not to represent an ANN, but to generate an ANN with arbitrary topology. This indirect encoding, coupled with NEAT was then used to form a new NE algorithm called HyperNEAT.

HyperNEAT begins with the development of Compositional Pattern Producing Networks (CPPN), described in Stanley (2007). A CPPN is almost identical to an ANN structurally, however it serves a distinguishing purpose. Like an ANN, they are a set of nodes connected to each other, with input, hidden, and output nodes. Unlike most ANN applications though, each node's activation function can be completely different, and this is what gives CPPNs their distinctive use. CPPNs have only a single output, and their inputs are mappings from Cartesian co-ordinates. This effectively makes the CPPN a function that relates two objects in space based on their relative positions. As shown by Figure 9, for a given coordinate, the outputs produced relative to another coordinate can form complex and interesting patterns.

This intuitively creates a function that has some ability to represent geometric patterns to some degree. This function is the cornerstone of HyperNEAT's indirect encoding. To understand how this function is used, we first describe another HyperNEAT concept - the substrate.

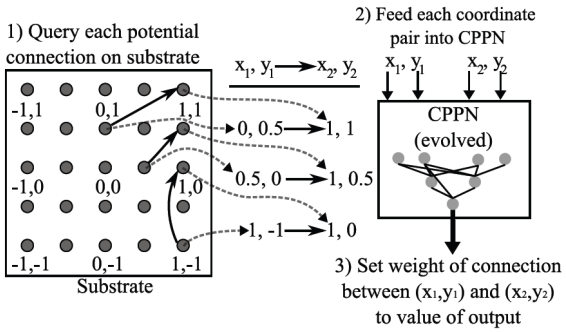

Figure 10: HyperNEAT Compositional Pattern Producing Networks (CPPN) determining substrate weights and connectivity (Stanley, D'Ambrosio, et al., 2009).

The substrate is also similar to an ANN, however each node in the ANN also has a geometric position. This position can be an arbitrary number of dimensions. Input nodes and output nodes can be placed according to some real-life geometric representation. For example, a car with radially attached sensors, would have a substrate with the input nodes mapped radially in the substrate. This way the geometry of the car is represented to some degree in the network. In theory this means that if more sensors were to be added to the car along the radius in arbitrary positions, the substrate would capture this information and this could be exploited to allow the sensors to work without further development.

The remaining piece is to connect the CPPN to the substrate, which is done simply by using each substrate node pair's coordinates as inputs for the CPPN and setting their weight, if the output is above a certain threshold, to the output. If the output is below the threshold then the nodes are not connected. It is also possible to manually specify connections in the substrate and simply set the connection weights of all connected nodes to the CPPN output. In theory this allows the CPPN to produce geometric patterns of connectivity in the substrate.

While the substrate itself is a function of hand crafting inputs and outputs as well as the CPPN, it remains to have a method for developing the CPPN itself. This is done with a modified version of NEAT, that has an extra mutation allowing a node to change its activation function. Through this it is then possible to evolve CPPNs and candidate solutions using a standard NE procedure. Each candidate solution is encoded as a CPPN. To evaluate the CPPN, an ANN is generated from a chosen substrate and the CPPN and evaluated for fitness. As usual, the fittest CPPNs are then modified and combined to form a new generation.

HyperNEAT presents a radically different solution to the previous NEAT and CNE approaches, with its indirect encoding. Not only does it allow the topology of the ANN to be changed post-evolution, but it theoretically allows for the geometry of the task to be exploited, for tasks where there is such a geometry, such as in robotics and controllers. This does come at a cost however, as the indirection added increases implementation complexity as well as search space complexity. Clune, Ofria, et al. (2009) showed that HyperNEAT can be extremely sensitive to geometric changes in the substrate, indicating that HyperNEAT may be difficult to apply well. Indeed, one of the biggest issues with applying NE methods is the vast number of hyper-parameters to be tweaked, and with the addition of a substrate, HyperNEAT only adds more. In this thesis we investigate and compare HyperNEAT with NEAT precisely in order to analyze this trade off when applied in intersection management.

We have so far described all the NE algorithms examined in this thesis. It is worth noting however, that other NE algorithms exist which are not included, such as SANE, which has been used successfully in robotics simulations (Moriarty and Mikkulainen, 1996; Moriarty and Miikkulainen, 1997). Various optimizations and alternatives exist for every step of the NE cycle, such as Covariance Matrix Adaptation (CMA-ES) (Floreano et al., 2008). These methods and techniques are excluded from this thesis firstly in order to limit scope of work, but also secondly because CNE, NEAT and HyperNEAT follows a particular path of development in the field of NE. We focus on these to understand how progressive generalization and complexification of the NE process has affected potential solution generation. We now examine cases where these methods were applied to other tasks of varying similarity to this thesis's task of controller development.

2.2.4 Neuro-Evolution Applications

We have covered the specifications for the various NE methods that are examined in this thesis. Now we will take a look at some of the research in which NE has been applied, in general as well as more specifically to tasks similar to intersection automation.

Dachwald (2005) made use of so called Evolutionary Neuro-Control to develop optimal interplanetary trajectories for low-thrust spacecraft. Despite the differing name, the algorithm described in the thesis can be considered a form of CNE. A feed forward network with hidden layers was evolved using an EA which then produced outputs for a given spacecraft in a simulation. Results in the thesis were good, with trajectories developed that significantly outperformed those developed by human experts.

Agapitos et al. (2007) made use of CNE to develop controllers for a simulated race car. The research itself compared this method with Genetic Programming, finding the CNE able to develop higher performing solutions given enough computational resources. Car controllers in the simulation are set up almost identically to the ones in this thesis, with cars having radially protruding distance sensors and speed and steering outputs. Both GP and CNE methods were able to successfully develop controllers capable of navigating multiple different race tracks effectively.

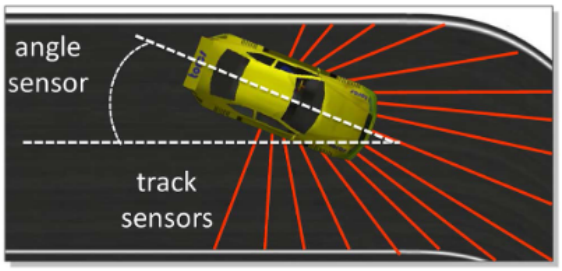

Figure 11: Sensor configuration used by the Neuro-Evolution of Augmenting Topologies (NEAT)-based entry into the 2009 Simulated Car Racing Championship (Loiacono et al., 2010; Cardamone et al., 2009). Each red line protruding from the car represents a distance sensor.

As previously seen, Kohl et al. (2006) made use of NEAT in developing a vehicle warning system, by using range finding and visual sensors. As presented by Loiacono et al. (2010), the 2009 Simulated Car Racing Championship included a NEAT-based car controller, modified slightly from Cardamone et al. (2009)'s proposed controller. As shown in Figure 11, the controller also made use of radially protruding sensors. Overall this controller achieved third place in the championship.

There are many other cases of NEAT applications, such as Willigen et al. (2013) evolving car controllers, and Duarte et al. (2016) developing robot swarm controllers. These theses all show the possibilities of NEAT as a tool for developing controllers for robots and cars. Duarte's swarm controller showed that NEAT can also be used in multi agent systems to produce co-operative behaviour. This multi-agent co-operative behaviour is critical to applying the above controller development strategies to intersection automation. Parker and Nitschke (2017) presented a set of controllers developed using NEAT that aimed to achieve intersection automation to some degree. The system developed in this thesis was able to outperform the above mentioned AIM system, and provides evidence on the viability of NEAT and NE as a whole for intersection automation. In the thesis however, controllers were developed for specific intersections, leaving open the question of how well these controllers would perform on unseen tasks. The next logical step would be to apply the same techniques but generalized to unseen intersections.

HyperNEAT has been applied successfully to many controller-development problems, some specifically to vehicle control. HyperNEAT's ability to make use of geometric information representative of the actual vehicles being controlled makes it ideal for such tasks. Clune, Beckmann, et al. (2009) and Yosinski et al. (2011) both made use of HyperNEAT to develop gaits for quadruped robots. In both cases, effective gaits were able to develop which outperformed hand-made gaits as well as methods based on NEAT. In Haasdijk et al. (2010), HyperNEAT's indirect encoding was used to develop controllers for modular robots that form organisms collectively. Since a given robot in this scenario could act as an arbitrary component of the whole organism, geometric information was able to be exploited without specific evolution being required.

While these theses have shown HyperNEAT to be highly effective in straightforward control problems, others have highlighted some limitations of the algorithm. Drchal et al. (2009) applied HyperNEAT to a car automation task, simulating traffic and optimizing for cars that were able to steer and avoid collisions. This may be seen as a potential implementation of intersection management, however the volume of cars in the simulation was very low (only five), and there was enough space for all cars to exist on the track without having to cross an intersection at the same time. The results show that HyperNEAT was able to effectively solve the task for the given vehicles, and that sensory input could be changed up to a point. At high resolutions however it was found that performance decreased when inputs where scaled. Hausknecht et al. (2014) on the other hand found that HyperNEAT was outperformed at compact state representations by direct encoding algorithms such as NEAT. They applied HyperNEAT to the problem of generalized video game playing. HyperNEAT was able to effectively perform in more complex state spaces. It is possible that the indirect encoding has a higher complexity which is not worth it when dealing with simple state spaces. In this thesis the state space is relatively simple, and so HyperNEAT's performance versus NEAT will aid understanding of when to use which method.

2.3 Conclusion

We have seen how computational methods have made incremental progress in the field of intersection management, from lane keeping through to data sharing traffic lights. This has reached its height with the work done on the AIM system, a heuristic based algorithm for allowing a much higher throughput than traditional human traffic through the use of centralized controllers. Finally we have seen how NE has been applied to the same space, with many self driving vehicle applications, vehicle avoidance systems, and multi-agent co-operative systems. We have also shown an array of NE algorithms that offer compelling possible methods for developing more advanced intersection automation systems.

The work in this thesis aims to expand upon this by contributing to the body of performance comparisons between the various NE methods referenced. Not only will a single instance of each algorithm be tested, but several factors which are common hyper-parameters to each. It also aims to expand on the implementations of simulated intersection automation systems with more generalized controllers and higher levels of traffic throughput.

Specifically, in the research theses mentioned in this background, several key areas are missing from current research. Firstly, a generalized intersection management controller has not been developed using NE which is intended to work on multiple different intersections. Secondly, while many iterative comparisons between CNE, NEAT and HyperNEAT exist, there are few that compare all three, as well as the evolution factors that affect performance. Deciding on hyper-parameters for evolution is still mostly a human decision based on intuition and guesswork, and has huge implications for computational costs. This thesis attempts to address these gaps by developing a general simulator and designing experiments which allow for comparisons across multiple factors. The simulator is able to control for all variables except those that are being tested, can handle multiple different types of controllers produced by CNE, NEAT and HyperNEAT algorithms, and can be adjusted with a set of factors that affect evolution. The simulator also is able to produce a baseline performance indication which can be used to benchmark the NE methods applied. The simulator is described in the following section.

Figure 12: The intersection simulator mid-run. The track is shown in gray, and the cars are shown as green rectangles. Padestrians are shown as small green circles.

3 Methods

At the core of this research is a custom built simulator as shown in Figure 12, meant to model a realistic traffic flow through an intersection. This simulator is the context in which all experimentation takes place, and allows for a fully controlled analysis of NE for intersection management specifically. The intent is for the model to function as an analogy for traffic problems that are faced in our current traffic system.

Of key importance is the ability of the simulator to test candidate controllers both in the evolution phase of experiments as well as afterwards in unseen tasks to determine and compare performance. To this end, each car's behaviour in the simulator is governed by a controller which can be configured as a parameter in the simulation. The model is designed as follows:

3.1 Overall Design

The model consists of a two dimensional space, filled with objects that represent traffic in an intersection. An intersection, cars and pedestrians exist in this space with co-ordinates and can interact with each other. When a simulation is run, it is fed in the structure of the intersection and a set of configuration options which detail the other parameters such as traffic, car controllers and simulation steps. With these parameters, the simulation then runs for the specified number of steps, generating traffic and simulating its course through the intersection. Throughout the simulation's execution, cars in the intersection control their speed and steering by using their controller. As the simulation executes, when cars or pedestrians come into contact with one another, it is considered a collision and both objects are removed from the model. The number of collisions that occurred during simulation between various objects is then returned, to be used as a metric for evaluation of the simulation.

3.2 Intersections

At the core of the simulation is an intersection, comprising a set of roads and traffic lanes. The intersection specifies paths that car and pedestrian objects travel along in the simulation, as well as points in the space where cars and pedestrians are generated periodically. Mathematically the intersection is modelled as follows:

Each simulation contains a single intersection. The intersection is modelled as a collection of directed, unweighted graphs, where vertices on the graphs are points in the two dimensional space. Graphs may contain cycles, and may have vertices with any size outdegree and indegree. Any vertex that has an indegree of zero is considered a root vertex, and all graphs must have at least one such vertex. This means that there can be no sections of the intersection in which there is no clear beginning to a path. Each graph in the intersection is designated as either a car path or pedestrian path exclusively, determining which type of traffic is generated on that particular graph. Cars and pedestrians that spawn in the simulation will begin at root vertices of their respective types of paths and travel through the graph from vertex to vertex, disappearing when they reach a vertex with zero outdegree. Pedestrians and cars never share the same path, however their paths will usually intersect at some point.

Lastly, edges between vertices can be flagged as inactive or active. In inactive edges, there are generally no crossing paths, and cars makes use of a heuristic controller designed to keep traffic flowing. This controller is described later in the chapter. Active edges are where custom controllers are used to control the cars' actions.

Below are the intersection designs used in the simulation (and their corresponding design in Figure 13), they are, in order from left to right and top to bottom:

- A large circle
- A small circle
- A crossing
- A standard four way intersection
- A standard four way intersection with two lanes
- A standard four way intersection with three lanes
- A road with a connecting on ramp
- A road with a connecting on ramp and a t-junction

3.3 Traffic Generation

Traffic in a simulation is controlled by a set of parameters which determine the shape of traffic, shown in Table 1.

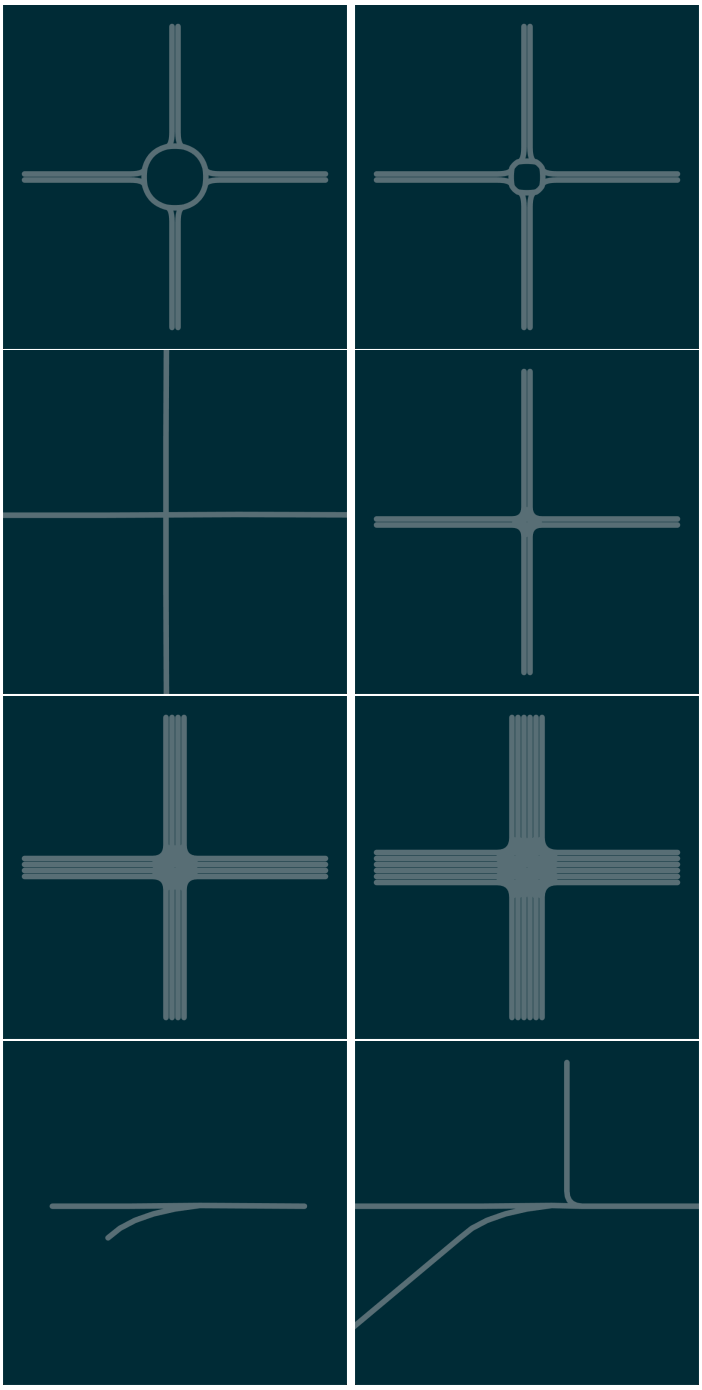

Figure 13: Intersection design used during controller evolution, as described in Chapter 3.2

Parameter	Description	Symbol
Min. Period	Minimum period between car spawns	$minP$
Max. Period	Maximum period between car spawns	$maxP$
Period Mult.	Function cycle multiplier	$mulP$
Randomness	Randomness factor for period	r
Pedestrian Period	Period for pedestrian spawns	$pedP$
Pedestrian Period	Randomness factor for pedestrian period	$pedR$

Table 1: Traffic generation parameters

These parameters are used to determine the rate at which traffic is generated in the intersection as a function of the number of steps elapsed, through Equation 1 (for cars) and Equation 2 (for pedestrians) where s is the number of steps elapsed and S is the total number of steps at which the simulation ends. The car function makes use of *sin* to produce traffic that mimics real world traffic with heavier and lighter periods in the day.

$$p = (minP + (1 - (0.5 + 0.5 * sin(mulP * PI * 2 * s/S))) * (maxP - minP)) * r \tag{1}$$

$$p = pedP * pedR \tag{2}$$

For a given rate p, cars spawn in the intersection at a random root vertex at that rate. If for some reason a car or pedestrian is present at a root vertex which is spawning a new object, it will be considered a collision as normal. This is integral to the design of the intersection, as it creates a limited amount of space in which cars can spawn. As will be discussed later in this chapter, traffic generally flows because cars are mostly using a heuristic controller which drives them forward. However, a controller that did not move much in the intersection in active areas would slowly cause traffic to backup on the intersection as heuristic controlled cars slowed down to match their speed. Eventually the traffic would back up all the way to the root node of the intersection graph, causing a collision each time a car spawned there. As long as there is a high enough volume of traffic that is spawned in a simulation, then because of the above mentioned effect, throughput is essentially necessitated by the simulation. If the throughput is not high enough, then the number of crashes will increase and thus measuring crashes is adequate for evaluating the controller and can be considered a proxy for throughput.

Pedestrians spawn following a similar scheme to cars, however they spawn far more sporadically with a greater randomness factor. Their behaviour is discussed later, however it is worth noting that they do not attempt to slow down at all, making it impossible for pedestrians to collide with each other. This is also important, as it means that collisions are only a function of car behaviour, instead of randomness introduced by pedestrians spawning.

We have described the layout of the paths in the simulation and how objects are populated during the simulation's execution. We now describe the actual mechanics of these objects and how they interact.

Measure	Simulator Value	Real World Equivalent
Car Radius	1	1 m
Car Top Speed	0.277	16.62 $m.s^{-1}$
Car Acceleration	0.0025	9 $m.s^{-2}$
Car Braking	0.004	14.4 $m.s^{-2}$
Pedestrian Radius	0.5	0.5 m
Pedestrian Speed	0.04	2.4 $m.s^{-1}$

Table 2: Various physics values in the simulator, with their coded values as well as real world equivalent values in units (as defined by the International System of Units)

3.4 Object Mechanics

An object in the simulation is something that has a two dimensional location as well as a radius that determines its area of collision. In this section we describe the basic principles behind how the objects move, changing their two dimensional location, and how they interact when they collide.

As the simulation executes, discrete steps are taken to update the status of all objects. For a given step, each object has a location, velocity and acceleration which is used to calculate its status in the next step. Each step also allows for controllers to influence various values for cars such as whether to accelerate or brake. Objects in the simulation are roughly modelled to have realistic sizes and speeds, and thus all values can be measured and converted into SI units. These exact values are shown in Table 2.

As objects move around the simulation, it is possible for them to collide, at which point they interact in various ways. Collision is detected between two objects using Equation 3 where *x1*, *y1* and *x2*, *y2* are the coordinates of the respective objects and *r1*, *r2* are their radii.

$$r1 + r2 > \sqrt{(x2 - x1)^2 + (y2 - y1)^2} \tag{3}$$

As mentioned, when cars and pedestrians collide with each other and between themselves they are removed from the simulation and a collision counter is incremented. Other than car and pedestrian collisions, cars are equipped with sensors which use collisions to feed information to the car's controller. These sensors and controllers are described later in the chapter, however they use the same mechanism for collision detection seen above.

While all objects adhere to the mechanics shown here, each object has specific internal behaviour and mechanics for adhering to them, which are described further.

3.4.1 Cars

Cars are the subject of interest in the experiment. They are represented in the simulation as a green rectangle, or purple when selected. Their behavior is determined by an ANN which uses sensor input on the car to determine how to accelerate and turn. This controllers is the independent variable in the experiment. Like all objects in the simulation, cars have a co-ordinate, radius and physics values that determine their movement. Cars spawn in the simulation from root vertices of the intersection and then begin following vertices in the graph. Once a vertex is reached, the car randomly decides on the next vertex to travel to from adjacent vertices. Once the car reaches a vertex with no adjacent vertices, it disappears and is considered a successful sample for throughput. Each car has two degrees of control, acceleration and turning, which are determined by evaluating the controller on the car. Acceleration controls the speed of the car within the limits of the physics of the simulation, as given in Table 2. Turning controls the path the car travels through the simulation, however this is only relative to the preset path the car is following along the intersection. Turning allows the car to veer off its path slightly while maintaining the cars trajectory. In order to reduce the amount of base behavior required from the controllers, and to speed up development, the cars are able to make use of a heuristic controller in certain parts of the intersection. The intersection is pre-specified with segments indicating which controller should be used and, generally, once the car enters into the intersection where there is a chance of collision then it switches to its ANN controller. Acceleration is completely controlled by the controller while turning is determined by the car's target vertex. Turning can then be overridden to some degree by the controller, allowing the car to evade other cars or obstacles in the simulation without veering too far off the track. The turning model is based on a real car having a wheelbase.

3.4.2 Pedestrians

Pedestrians are modelled similarly to cars, however they have a much smaller radius, lower speed and behave differently. They are represented in the simulation as small green circles. Unlike cars, pedestrians always act in a simple, predictable way and have no decision making process. They move at a constant rate in a straight line from the moment they spawn until a predefined finish. Pedestrians can collide with cars in the same way as other cars, which drives up the collision counter. Thus there is pressure for cars to learn to avoid pedestrians.

3.5 Controllers

A controller is a system that is used to process environmental input and produce output. The cars in this simulation obtain environmental input through range sensors radially attached to the car, and effect output by changing their speed and direction. In a simulation, a set of controllers is specified that each car will use. While each car uses the same set of controller designs, each car runs its own copy of the controllers, using its own sensory information and taking action based on the controllers' output. Which controller the car uses is dependent on where the car is in the intersection. In parts of the intersection that do not require co-ordination a simple heuristic controller is employed. The moment the car enters a section of the intersection that could potentially lead to collisions, an ANN based controller is used. The input for these controllers comes primarily in the form of range sensors, which are discussed next.

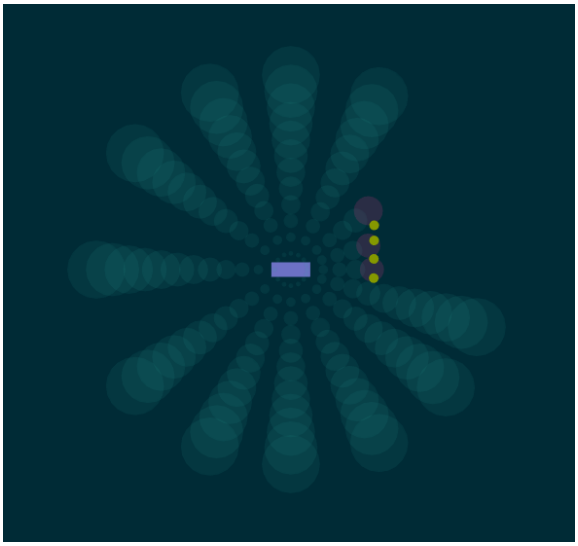

Figure 14: Car with sensors. The purple rectangle is the body of the car, and the light blue circles emanating from it are the sensors. Each sensor is made up of a column of increasingly large circles. To the top right, sensors detecting nearby objects in green are highlighted in purple.

3.5.1 Sensors

Each car is equipped with a set of range sensors that feed information into the car about how close objects are to the car. Each sensor is modeled as a set of circles arranged in a line perpendicular to the car's circle, with radius increasing the further away from the car. If a car is selected, its sensors can be seen as light blue circles, or purple if the sensor is activated. A sensor circle is considered triggered when it is in collision with another object, either a car or pedestrian. The sensor returns a value between 0 and 1, with 0 being no objects detected by the sensor, and 1 being an object triggering the closest sensor circle. Sensors that are farther away and also triggered are ignored, so that the maximum value of the triggered circles is taken. (In other words, the closest object's distance is detected by the sensor) As seen in Figure 14 most of the sensors would return 0, however the sensors directly in front of the car to the left will register a higher value.

Through these sensors, cars are able to detect objects in the immediate vicinity and react to them. Each car is equipped with fourteen sensors, giving a wide range of detection and the ability to react to objects before collision occurs.

Other types of sensors are possible and have been used in self-driving car applications. These include cameras and vision algorithms, radar, etc. In this thesis we focus on range sensors as they are prominent in the research this thesis is based on (Agapitos et al., 2007; Loiacono et al., 2010).

3.5.2 Heuristic Controller

When cars are in a part of the intersection that does not require co-ordination, they make use of a standard heuristic sensor, which has built in behaviour. The heuristic controller uses a single input: the speed of the closest car directly ahead of it. The output of this controller accelerates or decelerates the car to converge towards this speed so that the car matches the car in front of it and does not collide. The heuristic controller otherwise accelerates to full speed and makes no turning movements. By matching the speed of the car in front, pointless collisions are prevented and obvious heuristic behavior is achieved, such as cars lining up and waiting for the car in front during a traffic jam. Without this controller, the burden on the ANN controller to develop this behavior could increase evolution time and decrease performance.

3.5.3 ANN Controller

The ANN controller is the primary focus of each simulation, and is the context in which performance is measured. Each controller takes data from the car's range sensors and feeds it directly into the ANN's input layer. The ANN is then continually computed at each step, in order to determine two outputs: speed and turn angle. The car then applies these output values to it's physical controls. The topology and weighting of the ANN is completely predetermined before the simulation begins. In theory, as evolution iterates, the ANN controller should develop co-operative behaviour with high throughput, as continually higher performing controllers are selected and recombined each generation.

3.5.4 Controller Specifics

Each controller has a set of input sensors, as described above. These sensors are arranged as 14 proximity sensors arranged around the car, as shown in Figure 14. Each sensor provides a value between 0 and 1 that can then be fed into the neural network.

The outputs of the ANN are two values between 0 and 1, for speed and turn angle respectively. A value of 0 for speed indicates that the car should decelerate to a complete stop and a value of 1 indicates that the car should accelerate to full speed. A value of 0, 0.5 and 1 indicate, respectively, for the car to veer to the left as hard as possible, continue straight and veer right as hard as possible.

The internal structure of the ANN is dependent on the experiment being run and could be one of CNE, NEAT or HyperNEAT. In the context of the simulation the ANN is a black box and its workings are unimportant.

3.5.5 Limits of Controllers

Controllers are limited in how much they can affect a cars behaviour, in order to reduce erratic and unrealistic behaviour, as well as to prevent cars from straying too far off their course. The output of the controller is used to determine the car's intended speed, however the car will accelerate and decelerate to that speed at a predefined pace. The maximum speed is also preset, and cannot be influenced by the controller. Turning is even more restricted. Cars will follow the path of the intersection they are in and can only veer slightly based on controller output. Cars will never completely change direction or choose a new path based on the output of their controller.

3.6 Simulation Execution

According to the rules defined above, the simulation will execute for a set number of steps, recording collisions between cars and pedestrians, and controlling cars through ANNs that are being evaluated by the EA or tested in unseen tasks. Once the steps have finished, the simulation simply ends and the data is returned. Besides collisions, behavioural data is also returned, which will be analyzed in Chapter 5 and 6.

3.7 Trials

We have described the full inner workings of the simulation for its use in evolution and testing. In order to package this neatly for use, we define a trial. A trial is a single run of a simulation for a given set of parameters. The parameters must include an intersection, a car controller and settings that determine the type of traffic during the trial. Once the simulation has finished a set amount of steps, the trial will complete and the number of crashes that occurred during the simulation will be returned. All evolution and testing takes place by setting up parameters and then running a set of these trials.

3.8 Conclusion

The simulator defined above provides a powerful tool for evaluating and comparing a set of NE methods in a controlled manner. Realistic traffic conditions can be generated with rush hours, pedestrians, traffic jams and a high throughput of cars coordinating together. Trials provide a well defined unit of execution, which can be parametrized easily. In the next section we will describe how trials and the simulator are used to perform experiments from which data has been gathered.

4 Experiment Design

The above described simulation was used to execute a set of experiments in order to collect results. Broadly, these experiments examine several independent variables which control the simulation through above-mentioned hyperparameters such as traffic density and controller algorithm. The exact specification and structure of these experiments are described below.

4.1 Factors & Scenarios

In order to gather results, controllers must be developed and then tested. A set of hyper-parameters was chosen which represents a diverse range of different challenges and variables to be explored. These hyper-parameters are the independent variables and are called factors, and each set of factors is a scenario. Each of these scenarios is used to develop a single controller. There are three factors, each with three possible values, leading to 27 overall scenarios and controllers.

The first factor is the NE algorithm used for controller development. Its value can be CNE, NEAT or HyperNEAT. This factor affects the very nature of the controller itself, from network weightings to topology.

The second factor is trial difficulty which can be easy, medium or hard. This factor determines how dense traffic is during the simulation, with a higher difficulty resulting in denser traffic. In theory this should lead to a higher number of collisions, requiring more complex behavior in order to avoid them.

The third factor is problem exposure which can be one, two or three. This determines the number of different trials that will be used during evolution. A higher exposure will, in theory, increase the diversity of problems to solve and decrease overfitting.

The 27 scenarios allow us to study each factor's impact on the development of controllers for autonomous intersection management at several levels of granularity to determine if there are any strong correlations with controller performance.

4.2 Baseline

In order to provide a meaningful comparison and to establish difficulty, a baseline was used to adjust each scenario. This baseline was then also used during testing to compare and evaluate controller performance.

The baseline is setup as follows: a heuristic controller is used which sets the speed of the car to be maximum at all times with no steering control. This simulates a controller with no judgement or control, and crashes will be proportional to the amount of traffic in the simulation. Difficulty is defined as the average number of crashes that occur in a trial where the controller is one described above. The respective difficulties of each category (easy, medium and hard) are 5, 10 and 15. To achieve these specific difficulties, the parameters of each trial are tweaked. These factors affect the shape of traffic throughout the simulation, the intensity of this traffic and the number of pedestrians in the simulation.

The general complexity of the intersections themselves also play a factor. While it is difficult to measure complexity, it can be deduced by seeing how much more traffic is needed for a given intersection to get it to the same difficulty as another intersection.

All values throughout the rest of the thesis that represent collisions (fitness, testing performance, and so forth) will refer to a value that is normalized against the baseline. Therefore, a controller which scores, on average, 0.5 in a trial of hard difficulty is resulting in 7.5 collisions on average in that trial.

4.3 Controller Development

For a given scenario, NE was used to develop a controller using the factors of that scenario. For example, the scenario CNE-easy-two used the CNE algorithm to develop a controller, with trials that are of easy difficulty and with two unique trials. The EA hyper-parameters used are described in the next section. Table 3 shows the specific factor, scenario and trial configurations used. For each scenario, 10 iterations of the entire experiment were run, in order to reduce the impact of randomized initial conditions.

4.3.1 Evaluation

During NE, each candidate solution was subjected to a set of trials. In a trial, the candidate neural network is used to control every car in the intersection, although each car has their own copy. Throughout the simulation, collisions between cars are recorded and the final tally of these crashes constitutes the fitness function. Cars are constantly spawned on the track and move based on the heuristic controller, so there is pressure on the cars to complete their journey through the intersection, without needing a measure of throughput. Cars also cannot reverse in the intersection, so traffic will become backed up and collisions will start beginning at the start of the intersection if the cars perform poorly. Furthermore, each trial is repeated multiple times in order to increase the amount of randomness and diversity in challenge. This prevents lucky candidate solutions from getting a high fitness from a single trial that happens to be more forgiving.

Since the traffic is fixed and controlled but still randomized, the only possible good solution for candidates is to find a way to make it through the intersection at a good pace without crashing.

A candidate's score is finally the sum of collisions over all trials it is subjected to. The NE selects individuals in order to try minimize this score.

Scenario	Algorithm	Difficulty	Exposure	Trials
CxEx1	CNE	Easy	One	circle
CxEx2	CNE	Easy	Two	circle, on-ramp
CxEx3	CNE	Easy	Three	circle, on-ramp, crossing
CxMx1	CNE	Medium	One	four-way
CxMx2	CNE	Medium	Two	four-way, circle
CxMx3	CNE	Medium	Three	four-way, circle, on-ramp-extra
CxHx1	CNE	Hard	One	four-way-3-lane
CxHx2	CNE	Hard	Two	four-way-3-lane, four-way-2-lane
CxHx3	CNE	Hard	Three	four-way-3-lane, four-way-2-lane, circle-small
NxEx1	NEAT	Easy	One	circle
NxEx2	NEAT	Easy	Two	circle, on-ramp
NxEx3	NEAT	Easy	Three	circle, on-ramp, crossing
NxMx1	NEAT	Medium	One	four-way
NxMx2	NEAT	Medium	Two	four-way, circle
NxMx3	NEAT	Medium	Three	four-way, circle, on-ramp-extra
NxHx1	NEAT	Hard	One	four-way-3-lane
NxHx2	NEAT	Hard	Two	four-way-3-lane, four-way-2-lane
NxHx3	NEAT	Hard	Three	four-way-3-lane, four-way-2-lane, circle-small
HxEx1	HyperNEAT	Easy	One	circle
HxEx2	HyperNEAT	Easy	Two	circle, on-ramp
HxEx3	HyperNEAT	Easy	Three	circle, on-ramp, crossing
HxMx1	HyperNEAT	Medium	One	four-way
HxMx2	HyperNEAT	Medium	Two	four-way, circle
HxMx3	HyperNEAT	Medium	Three	four-way, circle, on-ramp-extra
HxHx1	HyperNEAT	Hard	One	four-way-3-lane
HxHx2	HyperNEAT	Hard	Two	four-way-3-lane, four-way-2-lane
HxHx3	HyperNEAT	Hard	Three	four-way-3-lane, four-way-2-lane, circle-small

Table 3: Overview of factors used in each scenario

Hyper-Parameter	Value
Iterations	100
Population	150
Trials	15
Mutation Rate (CNE)	0.4
Mutation Amount (CNE)	1
Add Node Rate (NEAT)	0.001
Add Link Rate (NEAT)	0.005
Remove Link (NEAT)	0.0005

Table 4: Hyperparameters for evolution

4.4 EA Hyper-Parameters

For these experiments, EA hyper-parameters were picked which allowed a large amount of exploration in each iteration, as well as a high number of iterations to allow as much development as possible. Table 8 shows the specific hyper-parameters used. The Java library Encog was used for its implementation of CNE, NEAT and HyperNEAT. In most cases, default Encog values were used. See Appendix A for these defaults. More specific settings for each algorithm are described below. For the trials variable, the set of trials in the scenario were repeated multiple times and the collisions summed, so that each candidate evaluation ran exactly 15 trials regardless of how many unique trials there were.

4.4.1 CNE

With CNE, there is no topological evolution, and thus the only parameters which are evolved are the weights of the network. The network itself must be preselected by hand. Care was given to select a network topology which was robust enough to encode complex potential solutions, while not being overly complex and difficult to evolve. This selection and selections for other parameters was made on the basis of background research which suggests these parameters to be reasonable for the given task. In future these parameters should be explored. For most applications a single hidden layer suffices.

In this case, the network topology consists of an input layer of 14 nodes to match the number of sensory inputs. Then a hidden layer of 30 nodes, fully connected to the input layer. Finally an output layer of 2 nodes, fully connected to the hidden layer. All nodes make use of the sigmoid function.

4.4.2 NEAT

NEAT is able to evolve the topology of a neural network during the genetic algorithm. This makes hyper-parameter tweaking minimal, with only some mutation types to set. Encog defaults were used except in the case of the AddNode and RemoveNode mutations, which were increased from a rate of 0.0000005 to 0.0005.

Figure 15: Three novel tracks used for testing the developed controllers

4.4.3 HyperNEAT

HyperNEAT hyper-parameters mainly consist of the substrate layout. The substrate used here mirrors the cars themselves as closely as possible, with the radial sensor inputs relating to radial inputs in the substrate. The acceleration and braking output as well as the steering output are located closer to the center, separated along the Z axis. No hidden nodes are used.

All weights in the network are then derived from the CPPN produced during evolution.

4.5 Testing

After evolution, the highest performing controller for each scenario's 10 repetitions was tested in order to determine its true performance on unseen tasks. The baseline heuristic controller was also added to the group of controllers for testing so that performance could be compared, resulting in 28 controllers overall.

Three novel tracks were used for testing, shown in Figure 15. They include two designs with complex intersections and one that is a simple straight line but which has many pedestrians crossing.

The tests, similarly to evolution, consisted of three difficulties that are equivalent to those in evolution, leading to nine distinct tests for each controller. Each test was then repeated 1000 times, giving 9000 total data points for each controller.

4.6 Conclusion

To summarize, three different evolution factors were explored. Each of these factors (algorithm, difficulty and exposure) have three possible values, leading to a total of 27 controllers being developed. These 27 controllers were developed by using the simulator to control for all other variables, except the ones of interest. A benchmark controller was also added which received no evolution and uses a basic heuristic function for controlling cars. Finally, these controllers were examined and tested on novel problems in order to analyze and evaluate them. In the next chapter we present the results from this examination and testing.

5 Results

We now present the results obtained during evolution and in testing. First we see the performance of the different scenarios in terms of fitness during evolution. This gives insight into how effectively the NE algorithms were able to search the problem space, and may reveal insights for understanding controller testing. Next we take a look at several metrics for evaluating the controllers, collected from the testing phase. This includes the performance in terms of crashes, the behaviour of the controllers and the structure of each controller's ANN. For performance, statistical tests are used to compare the number of crashes generated by each controller on a set of unseen trials. For behaviour, we look at the ANN outputs of the controllers as well as paths taken by cars in the simulation. For ANN structure we examine the network topology as well as analytical complexity of each network. These results will then be discussed and related to previous work in order to answer our research questions, in Chapter 6.

5.1 Evolution

The first entry point for analysis is to simply graph the fitness of the controllers during evolution. Figure 16 shows the average fitness during each experiment at each iteration of evolution.

Key takeaways from this figure are that fitness appears to improve dramatically in the first 20 iterations, and not much thereafter. With less iterations and a higher population size, evolution might prove to be more effective for a similar amount of computational power. It is also possible that the problem space is too difficult to search and is not continuous enough for NE to be effective. Also of note is that no single algorithm appears to be dominant. HyperNEAT reaches the best fitness six times, but only marginally, otherwise the fitness track ends in a similar result in most cases. Overall the path of evolution for each algorithm is similar, in terms of final fitness and even gradient.

It is also clear that there is some sensitivity to factors which affects fitness. While the controllers developed on easy experiments tend to reach a fitness around 0.5, medium and hard controllers reach around 0.75, except for *hard one* in which all algorithms reach a fitness close to 0.5. This may be a function of the track design and some inherent solvability.

In Figure 17 we see the average fitness distribution of the population in the last iteration of evolution for each controller. Again, most scenarios appear to evolve in a similar fashion with a distribution scoring between 0.3 and 1.2, except for a few exceptions. There is no clear factor which would appear to result in a higher fitness. Certain scenarios such as NxEx2 scoring poorly however point, again, to a sensitivity issue in evolution.

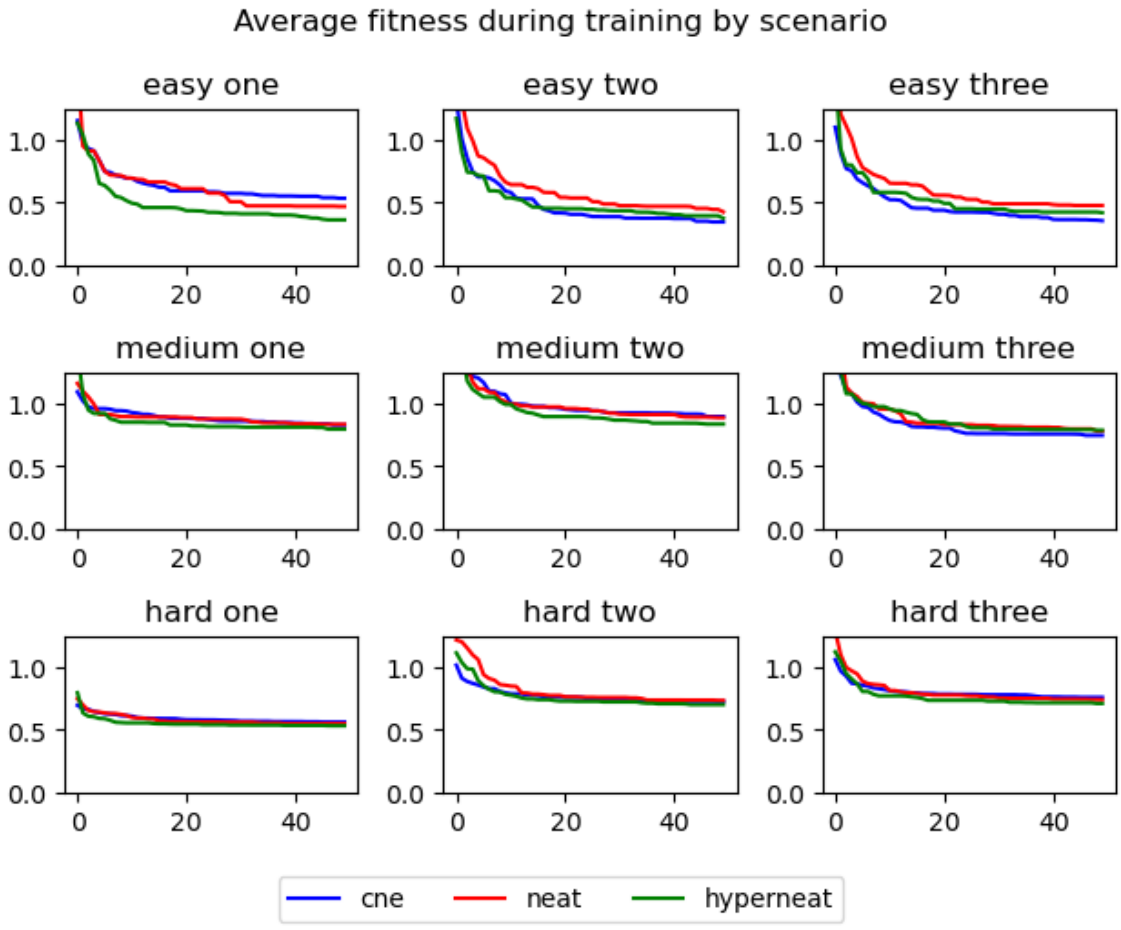

Figure 16: Average fitness as a function of evolution iterations for each algorithm in each scenario. All values are normalized to their respective difficulty's baseline.

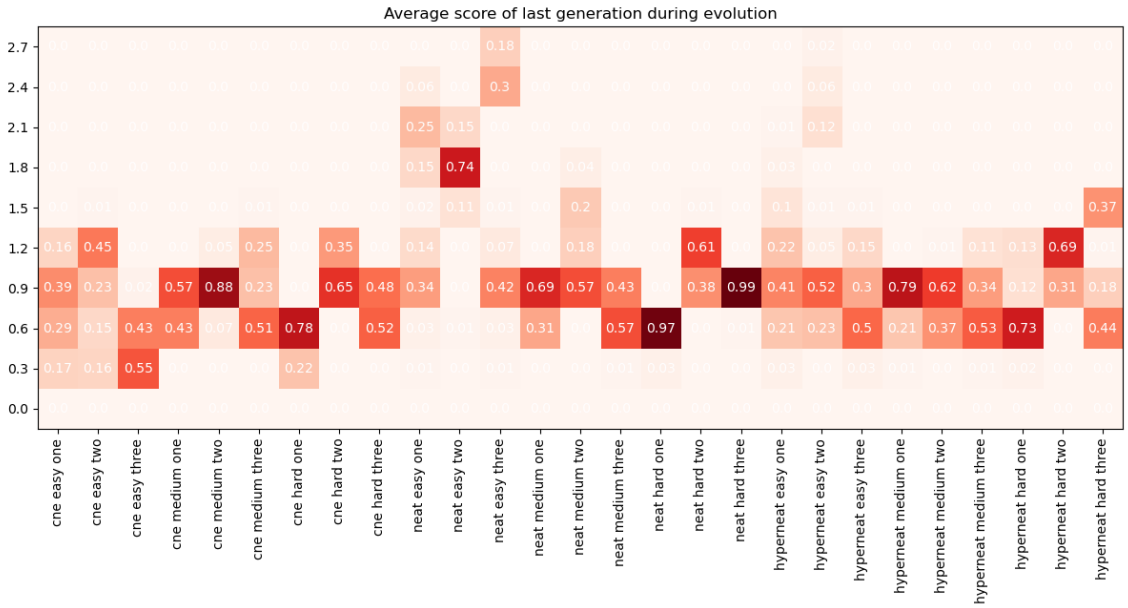

Figure 17: Distribution of fitness of the final generation at the end of evolution for each scenario

Referring now to Figure 23 from Appendix A, we examine how the evolutionary algorithms explored their problem space. A great spread across fitness for a given iteration would imply great exploration, as most solutions will be worse than the current best solution. It does appear that CNE has a slight edge in exploration throughout evolution, as shown by a grater variety of solutions being explored in the later iterations. NEAT and HyperNEAT on the other hand explore less as the iterations go by, but explore many solutions during the initial phase of evolution. This could have dramatic effects on the type of solutions discovered by each algorithm, as greater exposure initially could lead to more varied strategies. Exploration during later iterations will most likely result in small iterative improvements in strategy. These results are mirrored somewhat in Figure 16 where CNE is shown to have a smoother curve while NEAT and HyperNEAT have sudden sharp improvements.

Overall these results seem to indicate a similar ability for each algorithm to explore the problem space. While CNE explores a wider range initially, most likely due to its large network, it is still matched in fitness by the other algorithms, which require less hyper-parameter tuning.

5.2 Statistical Analysis

In this section we evaluate the testing results of the 28 controllers tested on nine unseen tests, repeated 1000 times. All data is aggregated and normalized to a baseline value, based on the performance of the baseline controller. Thus a value of 1.0 represents baseline performance, similar to a controller that has a single fixed output. Two levels of aggregations are used. The first is at the controller level: Each of the 28 controllers' testing data is aggregated to give 28 distributions of 9,000 data points. The second is at an experiment level: For each algorithm, difficulty and exposure, data is aggregated by the possible values of those factors to give 3 groups of 3 distributions each, with 81,000 points of data per distribution (the baseline distribution of 9,000 points is included as well).

5.2.1 Normality & Significance

In order to ensure that results are significant, a distribution and matching test must be used. We see from Figures 24,25,26,27 that our distributions appear to indeed be normal. It is worth noting that due to the nature of the simulation, and the low number of crashes possible in the simulations, the number of normalised values is low, meaning the distributions are not continuous. This makes normality testing using various methods impractical. Instead in the above mentioned figures we use histogram plots and probability plots to show that the distributions for every aggregation follow a normal shape, especially when plotting the probability distribution versus the normal distribution.

Given that we have established reasonably that our distributions are normal, we can apply the T-test to show where there is significance between our distributions. We use Welch's T-test (Welch, 1947) and apply Bonferroni correction (Bonferroni, 1936) to account for multiple comparisons. Tables 9,10 show the results of T-tests for each distribution, with comparisons highlighted in red where the null hypothesis cannot be rejected (the null hypothesis being that the samples are from the same distribution). It can be seen that in most cases results are statistically significant, with a few exceptions.

5.2.2 Results

The crux of this thesis is shown in Figures 18 and 19. Here we compare the raw performance of each controller when tested on unseen tasks. Performance is measured in exactly the same way as fitness, by collisions normalized to a baseline. A lower score indicates a better performance. Scores can be over the baseline, indicating that performance was worse than the baseline.

From Figure 18 we can see that the best performing controller was developed using NEAT, with a difficulty of easy and an exposure of three. This controller achieved the lowest mean, median and 25th percentile. On the other end, HyperNEAT, with a difficulty of hard and an exposure of three performed the worst. From comparisons with all other controllers, Table 9 shows these results to be statistically significant.

Just less than half the controllers performed higher than a basic heuristic benchmark, while several others outperformed the benchmark well, with the best controller being markedly improved over the benchmark, scoring 0.75 on median. Furthermore, various settings for each factor are scattered across the graph. This implies a high sensitivity to evolution hyper-parameters.

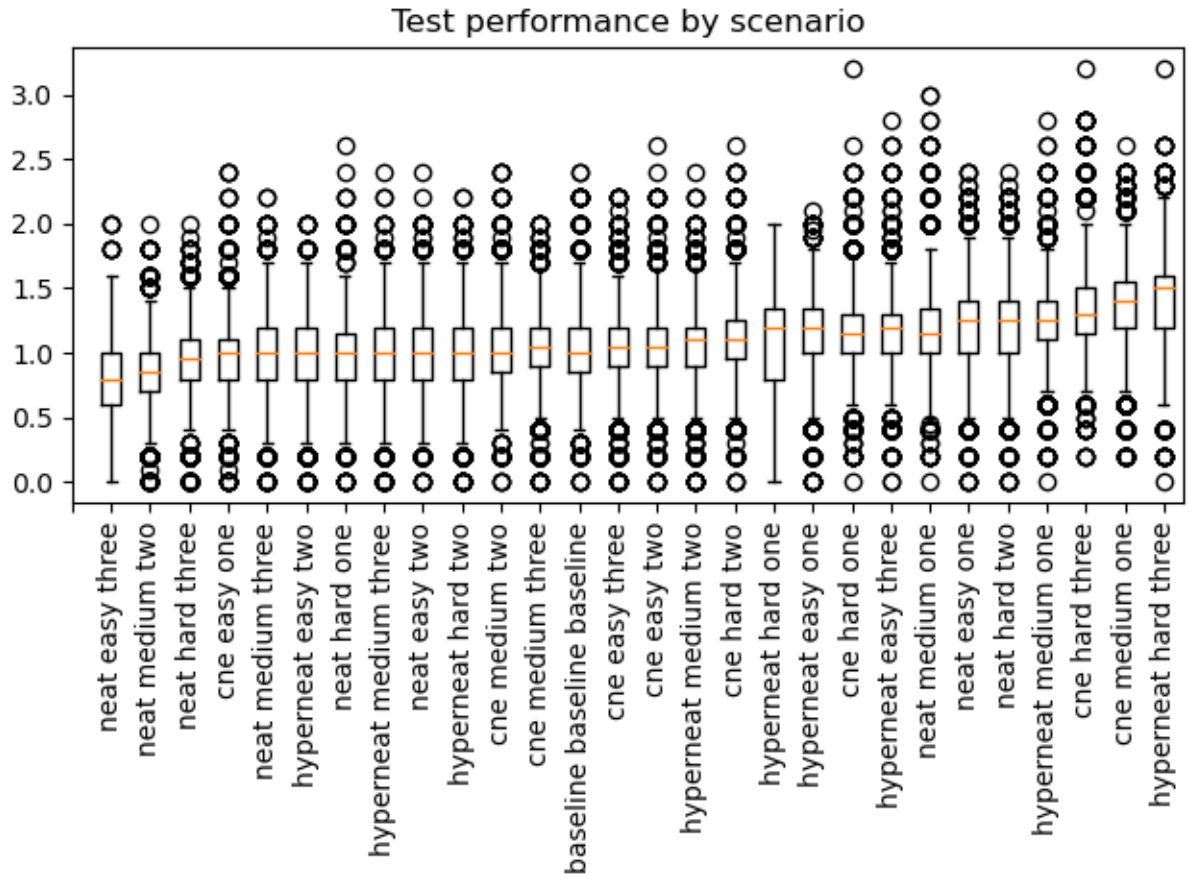

Figure 18: Performance of each controller during testing measured by average number of collisions, sorted by mean

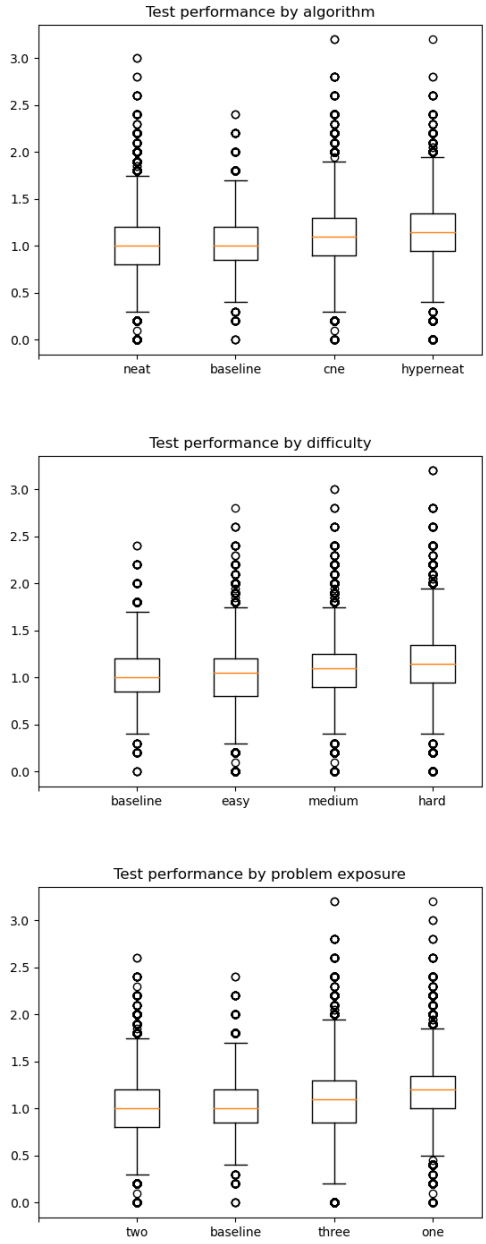

Figure 19: Performance of each controller aggregated by factor during testing, sorted by mean

From the factor comparison graphs in Figure 18 we can see further that overall factors have a low significance in performance. This further reinforces the evidence for sensitivity in NE. Problem exposure appears to benefit from a higher value, with the best controller being generated with exposure of three and the top 15 controllers only having 3 controllers developed using a problem exposure of 1. This benefit however does not follow a linear function with respect to exposure as two comes out as the best performing hyper-parameter. It does however, appear that there is a correlation between increased difficulty and poorer controllers. It seems that the cost to evolution difficulty does not pay off with more developed controllers.

Overall it appears that arbitrary scenarios may lead to poor or excellent controllers, and that the factors overall did not have clear correlations with performance. Some key combinations of factors however lead to exceptionally high, and low, performing controllers, indicating a high sensitivity for evolution. In contrast to Parker and Nitschke (2017), this low performance most likely derives from testing on multiple unseen trials. This could indicate that while specialized controllers are effective, they are unable to be used on intersections they are not evolved for. We now examine the behaviour of these networks in order to understand better why we see the differences we do.

5.3 Evolved Behaviour Analysis

We now refer to Figure 20 and 28 and examine the outputs produced by each controller during testing.

Our highest performing controller, NEAT easy three, has a very simple behaviour. It is mostly full speed the entire time, with 9% of the time a complete slowdown. In terms of steering there is a nearly even split between left and right.

Contrasted with the worst performing controller, which steers full to the left around 90% of the time and spends much more time fully slowed down, it would seem that a key factor in controller behaviour is ability to steer in different directions while maintaining as high a speed as possible. This is further supported by Figure 20 where high performing controllers like NxEx2, show similar patterns.

Figure 21 illustrates the behaviour of these controllers and reveals the differing performance using a simple experiment. While the baseline controller and HxHx3 cannot use evasive maneuvers, causing many crashes in the intersection, NxEx3 allows some cars to slow down and avoid collisions. This results in half the number of collisions. HxHx3 is seen to have some developed behaviour, in that it steers the car in the intersection, however it is not helpful in preventing collisions.

Also interesting to note is the much larger spread of behaviour seen in CNE controllers. This result is intuitive given how much larger and more connected the CNE controllers' networks are, however without any performance improvement this only makes the controllers' behaviour more unpredictable.

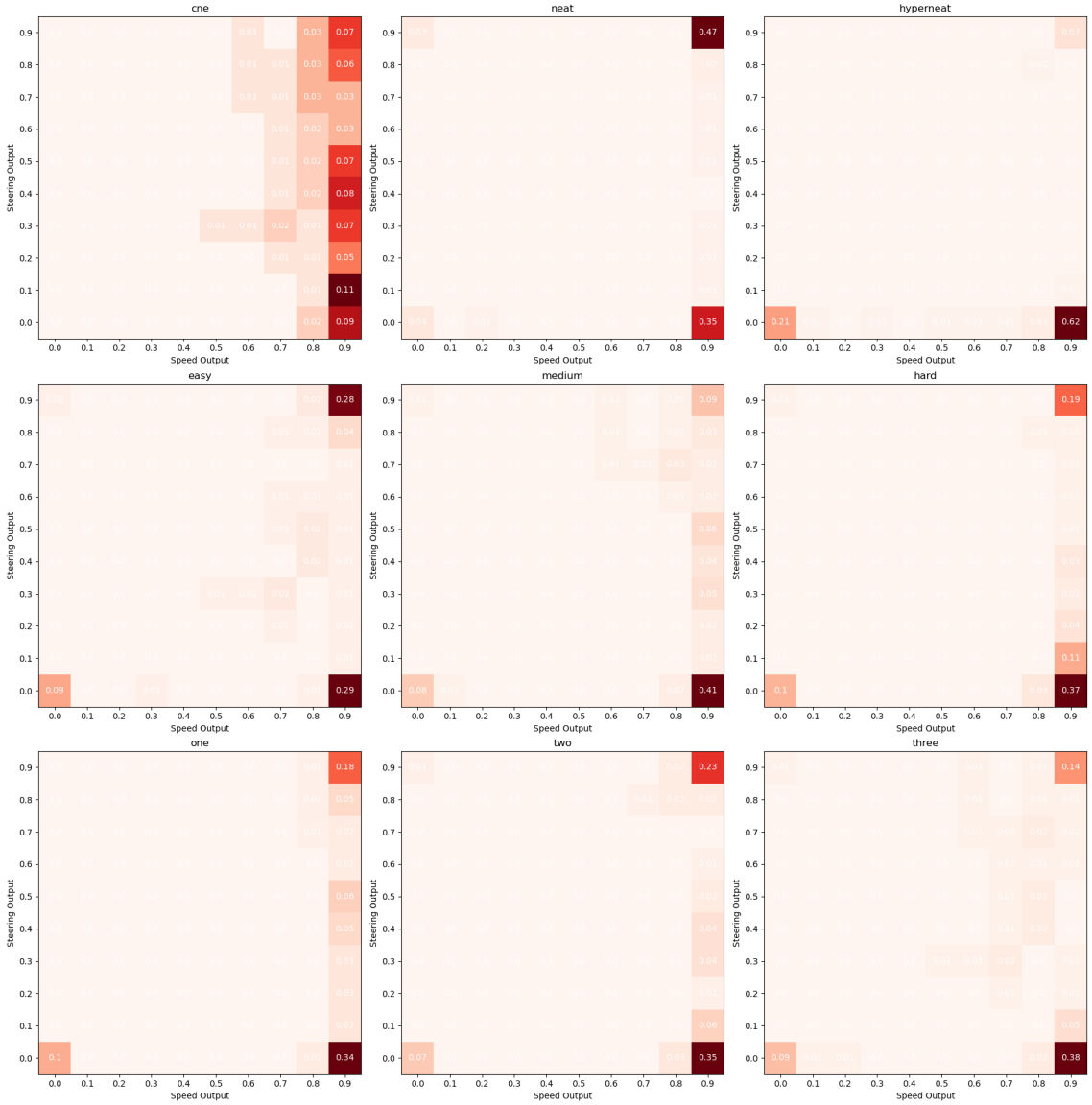

Figure 20: Behaviour of each controller aggregated by factor during testing. Each graph shows the distribution of outputs that cars exhibited on average over the time in the simulation.

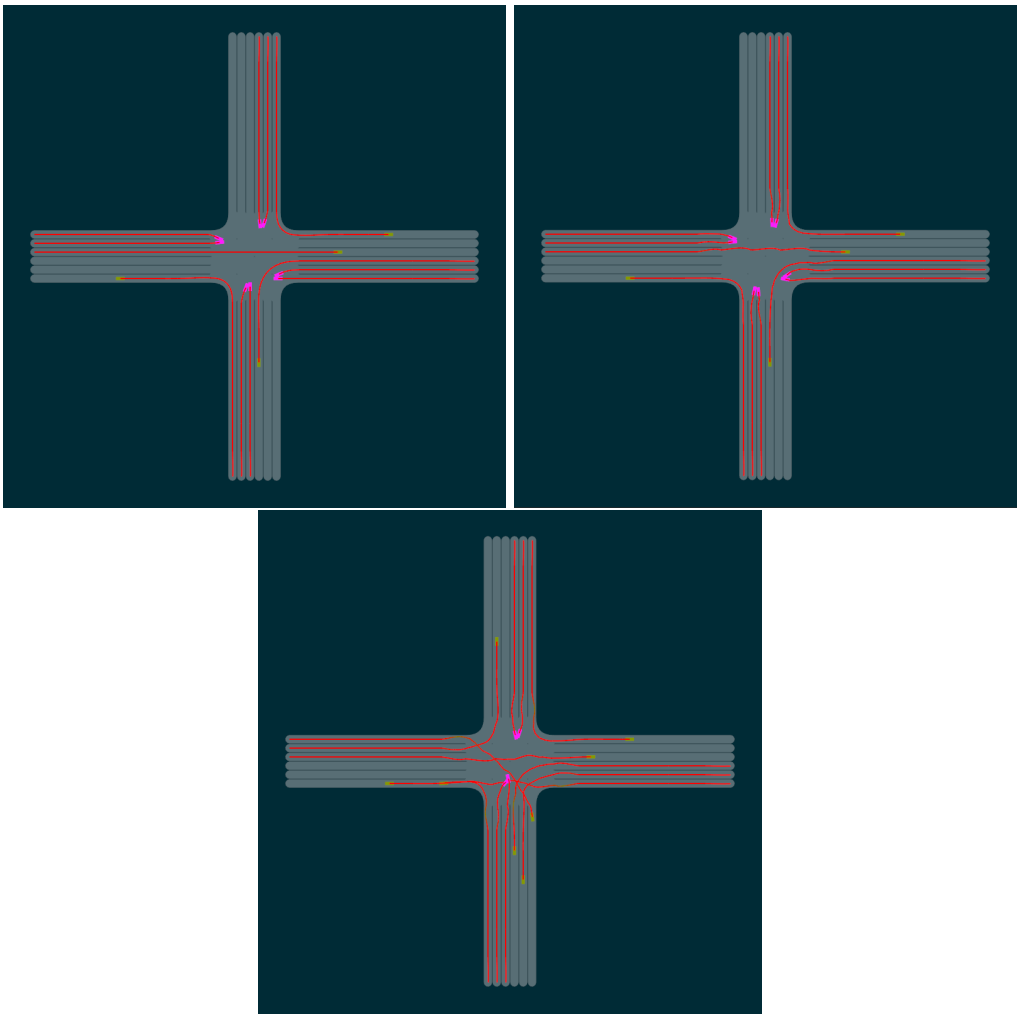

Figure 21: Paths followed by different controllers. Red line indicates a path that has been travelled. The color of this line also reflects the speed of the car by changing to the color green. Green cars were successful in avoiding collision while pink cars collided (and would normally be removed from the simulation). At the top left is the baseline controller, top right is HxHx3, the worst controller and finally on the bottom is NxEx3, the best controller.

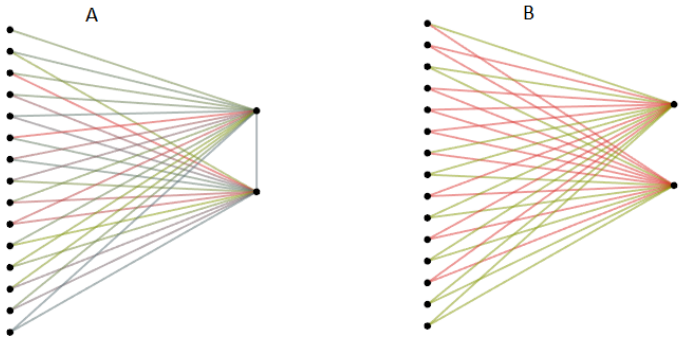

Figure 22: Artificial Neural Network (ANN) structure of the best controller (A, NxEx3) and the worst controller (B, HxHx3). Each black circle represents a node, and coloured lines between nodes indicate connections. A green connection indicates a positive weight, grey a weight close to 0, and red a negative weight. The nodes on the left most of the network are the input nodes, and calculations propagate to the right most of the network, which are the output nodes.

5.4 Evolved Neural Controller Complexity Analysis

Here we refer to Figure 29 and Figure 22 and examine the structure of each developed controller. The diagrams are interpreted as follows: Each black circle represents a node, with nodes vertically aligned being in the same layer. the far left layer is the input layer and the far right layer is the output layer. Connections are shown by lines drawn between nodes that range in color from red to grey to green. This color represents the weighting of the connection, with a brighter red indicating a more negative weight, and brighter green indicating a more positive weight.

As is expected, CNE structures are fixed and show the topology chosen during experiment design. Not much can be inferred from these diagrams other than the fact that the networks are indeed different. What is noticeable however is that many connections are weighted with a small value, as shown by being grey. This could explain the more spread behaviour pattern seen in Figure 20, as input nodes have smaller effects on the network.

Similarly HyperNEAT also has a topology fixed by the HyperNEAT substrate. The weights here however are extremely polarized, either being very negative or very positive. This may be the cause of the extremely modal behaviour seen in Figure 20 and would explain the low performance.

Finally in NEAT we see the most interesting topologies. Firstly, as seen in CNE, connections are much less polarized, with many more grey values. Secondly, there does not appear to be a correlation between network complexity and performance. The best performing controller only developed a single new connection, with a low weighting, between its output nodes. On the other hand, the most complex network, NEAT medium one, was one of the lowest performing controllers.

Controller	Complexity
NxEx1	49
NxEx2	50
NxEx3	45
NxMx1	54
NxMx2	48
NxMx3	44
NxHx1	44
NxHx2	45
NxHx3	48

Table 5: Complexity for each Neuro-Evolution of Augmenting Topologies (NEAT) controller.

To further formalize these results, we define a simple complexity metric, Equation 4, where Nc is the number of connections in the ANN and Nn is the number of nodes in the ANN.

$$C = Nc + Nn \tag{4}$$

Clearly, by design all CNE controllers have a complexity of 526, and all HyperNEAT controllers have a complexity of 44. Of interest are the NEAT controllers, which vary in complexity due to evolution. Table 5 shows each NEAT controller's complexity. Firstly, they are all far below the complexity of any CNE controller. Secondly, there is no clear correlation between higher complexity and performance. The best controller, NxEx3, has a complexity only 1 higher than the lowest, of 45.

It seems then that an important characteristic for performance is non-polarized connections between nodes, while network complexity makes far less difference.

5.5 Conclusion

We have presented the data and analysis produced by the experiments described in Chapter 4. This includes examination of the various factors' effects on evolution and fitness, the performance of the developed controllers on unseen tasks, and some intrinsic properties of the controllers themselves. In the next section we will discuss the important takeaways from these results and put them in the context of the thesis' research goals.

6 Discussion

We have presented the results from our data collection during both evolution and testing. We now examine and discuss how these results relate to the research goal of this thesis - that is, to compare sets of NE hyper-parameters to determine which ones lead to the best task performance, and to understand how these hyper-parameters affect task performance.

In terms of the evolutionary phase of the experiment, we are less interested in performance given that we see no correlation between evolution performance and test performance. This lack of correlation however, is interesting to note. The discussion around this is based on results presented in Chapter 5.1. There are clear optimizations to be made from the evolutionary phase though, which consumes an enormous amount of resources. As can be seen in Figure 16, performance does not increase much after 20 generations in any hyper-parameter scheme, which signals that much of the resources spent during evolution were wasted. There are several possible ways to improve on this. Most simply, decreasing evolution iterations and increasing population size may result in greater exploration of the problem space. More complex evolutionary systems may also help produce better evolution, as in this case the trade off of per-generation complexity for more powerful exploration should pay off. It is also possible that the problem space is far too difficult to search and requires enormously more resources than were used in our experiments. From Figure 17 we can see that the distribution of fitness in the final generation of evolution for each controller is similar across the board, except for one or two controllers. In most cases the distribution centers around the 0.9 score mark, further giving evidence that the problem space is difficult to search and may simply require more resources than were used. Overall, analysis of the evolutionary phase did not provide us with any significant information towards answering the primary research question of this thesis, that is - which hyper-parameters produce the controllers with the highest task performance?

Test performance results given in Chapter 5.2.2, on the other hand, were very revealing. In terms of our three factors, a clear best combination emerged with the NxEx3 controller (refer to Table 3 for the definition of NxEx3). This controller had a score significantly better than the baseline, achieving a median score of 0.795 across all test tasks. Furthermore a perfect score of 0 fell within the 25th percentile of its performance distribution across all test tasks. As shown in Figures 20 and 21, the controller developed behaviour for navigating through intersections and avoiding collisions, by slowing down when approaching the intersection to avoid collisions, and steering mostly straight with some variation. This behaviour intuitively leads to a high fitness, as throughput is kept high by cars only slowing down when behind other cars they may crash into. Having steering output split evenly between left and right would imply that this controller also has a greater range of movement and actions it can take in the intersection. We could then say that this combination of hyper-parameters leads to the best task performance, with statistical significance. While this answers our primary research question, we are still interested in why this particular combination works best and how each hyper-parameter contributed to the controller's performance. In the following chapters we discuss the hyper-parameters themselves in further detail.

While some controllers achieved a high task performance, such as NxEx3, NxMx2, NxHx3 and CxEx1, many others failed to improve on the baseline. Among the hyper-parameters, there were no major performance differences on average or clear correlations. This is most clearly shown in Figure 19 where aggregating with each hyper-parameter revealed very similar results across the board. This effect in individual controllers can be seen when one fixes two of the three hyper-parameters and then notes the order in which the values of the final hyper-parameter rank in performance. For example, with a fixed factor of NEAT and easy, different exposures lead to controllers with medians of 1.25, 1, 0.75 (respective to one, two, three exposure) where as fixing for HyperNEAT and easy, leads to 1.2, 1, 1.2 (again, respective to exposure). It seems then that particular combinations of hyper-parameters are responsible for drastic performance differences. NxEx3, the highest performing controller, made use of hyper-parameters easy and three exposure, which when aggregated, where some of the worst performing. We have seen in previous work that NE algorithms can be sensitive to hyper-parameters (Clune, Ofria, et al., 2009), and here it seems that this is indeed the case as well in this problem space.

There are some clearer takeaways however. While NEAT controllers do appear towards the lower end of performance, as seen in Figure 18, the top three controllers all made use of NEAT, and NEAT was still the highest performing algorithm overall. We speculate that NEATs initially simplistic network and direct encoding provide the most efficient algorithm of the three in this particular task. HyperNEAT also begins with a simplistic network, but its indirect encoding may not be effective when the morphology of the controllers is static. The difficulty factor was revealed to result in lower performance the higher the difficulty. Given that higher difficulties require denser traffic and more computational resources, it is clear then that difficulty should be minimized in evolution. If, during evolution, the maximum fitness is achieved (as was not in this case) then it may make sense to increase the difficulty to allow more room for optimization. Doing this prematurely when maximum fitness is not achieved however would seem to be a waste of computational resources. A better solution may be to use transfer learning (Torrey and Shavlik, 2010) and progressively increase difficulty as maximum fitness is obtained, in separate evolution runs. When examining problem exposure we see that there is a measurable difference between controllers developed with an exposure to one trial vs those developed with multiple trials. Previous work using NE to develop controllers such as Parker and Nitschke (2017) which exposed the controller to a single problem may find that their developed controllers are not able to adapt to unseen tasks as well as if they had been exposed to more tasks. While the intuitive result that evolving with multiple tasks yields controllers that are more adaptable, the question of exactly how many tasks is less obvious to answer. In our experiments, the middle ground exposure to two tasks seemed to be best overall, however the best performing controller was exposed to three different tasks. It is difficult to extrapolate how further increasing problem exposure would affect performance, but perhaps a similar technique as mentioned above with transfer learning between increasingly more varied tasks may work well to develop a generalized controller.

Finally, some insights are provided by the topology and weighting in the evolved controllers, as seen in Chapter 5.4. Network size or depth were clearly not limiting factors during evolution, as the best NEAT controllers were barely more complex than their initial topology. These results are supported by similar work in Huang and Nitschke (2020). Furthermore these NEAT controllers tended to outperform the CNE controllers, which were using multiple layer networks with many more hidden nodes per layer. What may account for this is the non-polarized weightings as seen in Chapter 5 which were prevalent in NEAT networks. The more polarized and extreme weights in HyperNEAT and CNE networks may lead to more extreme behaviour that may overcompensate when only subtle adjustments to steering and acceleration are needed. This would also explain the behaviour seen in Figures 20 and 21.

6.1 Conclusion

Our collected results give many insights into the various hyper-parameters that go into our NE scheme. Some key hyper-parameters were identified as having little impact on task performance, namely difficulty and problem exposure, and some as having some impact, such as algorithm. Many hyper-parameters were identified as not having any impact, such as network complexity and high iteration. Particular combinations of hyper-parameters seemed to yield effective controllers that showed interesting and emergent behaviour, however without strong patterns throughout controllers when using these hyper-parameters, it would seem that these NE algorithms are prone to sensitivity problems. The answer to our research questions is that the particular combination of NxEx3 produced the best controller, due to using the strongest algorithm, and using a problem exposure of more than one. For the algorithm hyper-parameter, NEAT slightly edges out CNE and HyperNEAT in performance. Problem exposure and difficult proved less definitive, but clear benefits for problem exposure above one was shown, and minimizing the computationally expensive difficulty may speed up evolution in future.

7 Conclusion

We have investigated many factors relating to the evolution and performance of NE controllers for intersection management. First, as an overarching result, it would appear that evolving controllers can be effective. Within a reasonable time frame, controllers which could substantially improve on the baseline were developed. While high performing controllers were evolved, without clear performance correlations for any hyper-parameter it remains difficult to tell which combination of hyper-parameters to use during evolution to achieve this performance. This was seen by the spread of factor hyper-parameters across controller rankings, as well as a high number of controllers failing to beat the baseline heuristic. Our results show that there is perhaps a large gap still between previous implementations and a fully working, robust, general solution to intersection management based on Neuro-Evolution (NE).

In spite of this, the results clearly show numerous ways to improve evolution in future research. Given that improvements during evolution appear mostly during the first 20 generations, improvements could be made by increasing the population size and decreasing the number of iterations. This would increase explorative power without needing more computational resources.

It is also clear that when the problem is not being fully solved, increasing the difficulty of evolution only stymies the controllers' development. Instead a far more productive hyper-parameter to increase would be the problem exposure during evolution. Higher problem exposure greatly benefited controllers during testing on unseen tasks, suggesting that the controllers were less overfit and more generalized to the overall task.

In terms of complexity, results showed that complexity was not correlated with performance in any way. Simple NEAT controllers outperformed much more complex Conventional Neuro-Evolution (CNE) controllers, and did not evolve a higher complexity even given the capability.

Lastly, NEAT emerged as an out performer, due to its non-polarized weightings and simple, predictable behaviour that was concentrated in the most effective places. Neuro-Evolution of Augmenting Topologies' (NEAT) evolving topology also made for the simplest hyper-parameter tweaking among the three NE algorithms. The highest performing controller was produced with NEAT and the exact set of hyper-parameters used for this controller was NxEx3.

7.1 Future Work

While many hyper-parameters have been explored in this thesis, there are many more possible configurations. While difficulty does not seem to have much potential for further exploration, increasing problem exposure to more trials could potentially benefit controllers greatly. Tweaking the topology of the HyperNEAT substrate and the CNE network could also increase performance. We now can see that a complex network is not necessarily needed for high performance, and that interesting connections such as those between output nodes can lead to desired behaviour.

Other ANN structures could also be investigated, such as recursive layers in Conventional Neuro-Evolution, or Long Short-Term Memory structures. The structures in this thesis are limited to conventional feed forward networks and some limited recursion possible in NEAT.

Results showed that a particular set of factors (NxEx3 and HxHx3) led to extreme cases of performance during testing. Factors as a whole however did not show significant differences in performance. This indication of sensitivity is import to future work in NE, and should be investigated further. This could be done either through testing a wider spectrum of factors during evolution for the same problem space, or replicating the experiments presented in this thesis but with a different problem space. It would then be possible to determine if the factors which led to extreme performance where arbitrary or whether there is correlation between performance and factors not seen in our results.

It would appear that even with recent increases in computational power, these resources are a limiting factor in NE research. Statistical robustness as well as evolution performance could be improved by simply using a larger amount of computer resources and increasing the number of evolution repetitions, evolution iterations, evolution population, and so on. This may be the single factor allowing for the development of effective, generalized intersection automation through NE-developed controllers.

Acknowledgements

First and foremost I'd like to thank the academic institutions which made this thesis possible at all, namely my funder the National Research Foundation, the University of Cape Town Computer Science department, and of course my supervisor Geoff Nitschke, who has helped me shape my bare code into a fully formed piece of research.

I'd also like to thank my family for their support as well as advice on the world of academics, in which they have many years of experience.

Finally I'd like to thank my girlfriend Ailsa, as well as my close friends for helping me get through the difficult parts of this project, for providing feedback and their thoughts and for filling the gaps in my knowledge.

It has been an incredibly engaging and fulfilling journey to complete this thesis over a time in which my life has changed in so many ways, and I look forward to taking the lessons learnt with me through the rest of my life. Thank you.

References

Bonferroni, Carlo (1936). "Teoria statistica delle classi e calcolo delle probabilita". In: *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze* 8, pp. 3–62.

McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.

Welch, Bernard L (1947). "The generalization ofstudent's' problem when several different population variances are involved". In: *Biometrika* 34.1/2, pp. 28–35.

Holland, John (1975). "Adaptation in natural and artificial systems: an introductory analysis with application to biology". In: *Control and artificial intelligence*.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". In: *nature* 323.6088, pp. 533–536.

Whiteley, D (1988). "Applying genetic algorithms to neural network problems." In: *Neural Networks* 1, p. 230.

Montana, David J and Lawrence Davis (1989). "Training feedforward neural networks using genetic algorithms." In: *IJCAI*. Vol. 89, pp. 762–767.

Maggiora, Gerald M, David W Elrod, and Robert G Trenary (1992). "Computational neural networks as model-free mapping devices". In: *Journal of chemical information and computer sciences* 32.6, pp. 732–741.

Pomerleau, Dean A (1992). *Neural network perception for mobile robot guidance*. Vol. 239. Springer Science & Business Media.

Whitley, Darrell et al. (1993). "Genetic reinforcement learning for neurocontrol problems". In: *Machine Learning* 13.2-3, pp. 259–284.

Whitley, Darrell (1994). "A genetic algorithm tutorial". In: *Statistics and computing* 4.2, pp. 65–85.

Moriarty, David E and Risto Mikkulainen (1996). "Efficient reinforcement learning through symbiotic evolution". In: *Machine learning* 22.1-3, pp. 11–32.

Alvarez, Luis, Roberto Horowitz, and Perry Li (1997). "Traffic flow control in automated highway systems". In: *IFAC Proceedings Volumes* 30.8, pp. 65–70.

Gomez, Faustino and Risto Miikkulainen (1997). "Incremental evolution of complex general behavior". In: *Adaptive Behavior* 5.3-4, pp. 317–342.

Hatipolglu, C, Keith Redmill, and Umit Ozguner (1997). "Steering and lane change: A working system". In: *Proceedings of Conference on Intelligent Transportation Systems*. IEEE, pp. 272–277.

Moriarty, David E and Risto Miikkulainen (1997). "Forming neural networks through efficient and adaptive coevolution". In: *Evolutionary computation* 5.4, pp. 373–399.

Naumann, R et al. (1997). "Validation and simulation of a decentralized intersection collision avoidance algorithm". In: *Proceedings of Conference on Intelligent Transportation Systems*. IEEE, pp. 818–823.

Svozil, Daniel, Vladimir Kvasnicka, and Jiri Pospichal (1997). "Introduction to multi-layer feed-forward neural networks". In: *Chemometrics and intelligent laboratory systems* 39.1, pp. 43–62.

Willis, M-J et al. (1997). "Genetic programming: An introduction and survey of applications". In: *Second international conference on genetic algorithms in engineering systems: innovations and applications*. IET, pp. 314–319.

Moriarty, David E and Pat Langley (1998). "Learning cooperative lane selection strategies for highways". In: *AAAI/IAAI* 1998, pp. 684–691.

Reynolds, Craig W (1999). "Steering behaviors for autonomous characters". In: *Game developers conference*. Vol. 1999. Citeseer, pp. 763–782.

Roozemond, Danko A (1999). "Using autonomous intelligent agents for urban traffic control systems". In: *Proceedings of the 6th World Congress on Intelligent Transport Systems*. Citeseer.

Stanley, Kenneth O and Risto Miikkulainen (2002). "Evolving neural networks through augmenting topologies". In: *Evolutionary computation* 10.2, pp. 99–127.

Abdulhai, Baher, Rob Pringle, and Grigoris J Karakoulas (2003). "Reinforcement learning for true adaptive traffic signal control". In: *Journal of Transportation Engineering* 129.3, pp. 278–285.

Dresner, Kurt and Peter Stone (2004). "Multiagent traffic management: A reservation-based intersection control mechanism". In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 530–537.

Haykin, Simon (2004). "A comprehensive foundation". In: *Neural networks* 2.2004, p. 41.

Lindner, Frank, Ulrich Kressel, and Stephan Kaelberer (2004). "Robust recognition of traffic signals". In: *IEEE Intelligent Vehicles Symposium, 2004*. IEEE, pp. 49–53.

Bazzan, Ana LC (2005). "A distributed approach for coordination of traffic signal agents". In: *Autonomous Agents and Multi-Agent Systems* 10.2, pp. 131–164.

Dachwald, Bernd (2005). "Optimization of very-low-thrust trajectories using evolutionary neurocontrol". In: *Acta Astronautica* 57.2-8, pp. 175–185.

Hallé, Simon and Brahim Chaib-draa (2005). "A collaborative driving system based on multiagent modelling and simulations". In: *Transportation Research Part C: Emerging Technologies* 13.4, pp. 320–345.

Kohl, Nate et al. (2006). "Evolving a real-world vehicle warning system". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 1681–1688.

Agapitos, Alexandros, Julian Togelius, and Simon Mark Lucas (2007). "Evolving controllers for simulated car racing using object oriented genetic programming". In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, pp. 1543–1550.

Stanley, Kenneth O (2007). "Compositional pattern producing networks: A novel abstraction of development". In: *Genetic programming and evolvable machines* 8.2, pp. 131–162.

Dresner, Kurt and Peter Stone (2008). "A multiagent approach to autonomous intersection management". In: *Journal of artificial intelligence research* 31, pp. 591–656.

Floreano, Dario, Peter Dürr, and Claudio Mattiussi (2008). "Neuroevolution: from architectures to learning". In: *Evolutionary Intelligence* 1.1, pp. 47–62.

Cardamone, Luigi, Daniele Loiacono, and Pier Luca Lanzi (2009). "Evolving competitive car controllers for racing games with neuroevolution". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 1179–1186.

Clune, Jeff, Benjamin E Beckmann, et al. (2009). "Evolving coordinated quadruped gaits with the HyperNEAT generative encoding". In: *2009 iEEE congress on evolutionary computation*. IEEE, pp. 2764–2771.

Clune, Jeff, Charles Ofria, and Robert T Pennock (2009). "The sensitivity of HyperNEAT to different geometric representations of a problem". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 675–682.

Drchal, Jan, Jan Koutnık, and Miroslav Snorek (2009). "HyperNEAT controlled robots learn how to drive on roads in simulated environment". In: *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, pp. 1087–1092.

Stanley, Kenneth O, David B D'Ambrosio, and Jason Gauci (2009). "A hypercube-based encoding for evolving large-scale neural networks". In: *Artificial life* 15.2, pp. 185–212.

Haasdijk, Evert, Andrei A Rusu, and AE Eiben (2010). "HyperNEAT for locomotion control in modular robots". In: *International Conference on Evolvable Systems*. Springer, pp. 169–180.

Loiacono, Daniele et al. (2010). "The 2009 simulated car racing championship". In: *IEEE Transactions on Computational Intelligence and AI in Games* 2.2, pp. 131–147.

Salichon, Max and Kagan Tumer (2010). "A neuro-evolutionary approach to micro aerial vehicle control". In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, pp. 1123–1130.

Torrey, Lisa and Jude Shavlik (2010). "Transfer learning". In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, pp. 242–264.

Yosinski, Jason et al. (2011). "Evolving robot gaits in hardware: the HyperNEAT generative encoding vs. parameter optimization." In: *ECAL*, pp. 890–897.

Gershenson, Carlos and David A Rosenblueth (2012). "Self-organizing traffic lights at multiple-street intersections". In: *Complexity* 17.4, pp. 23–39.

Cools, Seung-Bae, Carlos Gershenson, and Bart D'Hooghe (2013). "Self-organizing traffic lights: A realistic simulation". In: *Advances in applied self-organizing systems*. Springer, pp. 45–55.

Willigen, Willem H van, Evert Haasdijk, and Leon JHM Kester (2013). "Evolving intelligent vehicle control using multi-objective neat". In: *Computational Intelligence in Vehicles and Transportation Systems (CIVTS), 2013 IEEE Symposium on*. IEEE, pp. 9–15.

Hausknecht, Matthew et al. (2014). "A neuroevolution approach to general atari game playing". In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4, pp. 355–366.

Duarte, Miguel et al. (2016). "Evolution of collective behaviors for a real swarm of aquatic surface robots". In: *PloS one* 11.3, e0151834.

Parker, Aashiq and Geoff Nitschke (2017). "Autonomous intersection driving with neuro-evolution". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, pp. 133–134.

Huang, Allen and Geoff Nitschke (2020). "Evolutionary Automation of Coordinated Autonomous Vehicles". In: *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE CEC.

A Method Parameters

See https://github.com/MPCherry/MastersCode for source code of the method and experiment design.

Parameter	Value
Mutate Chance	0.4
Mutate Type	Weight Perturb
Mutate Amount	1
Crossover Chance	0.8
Crossover Length	10
Selection Strategy	Tournament
Selection Rounds	4
Elite Rate	0.3

Table 6: Evolution parameters for CNE

Parameter	Value
Mutate Chance	0.1125
Mutate Type	Weight Perturb
Mutate Amount	1
NEAT Crossover Rate	0.5
Add Node Chance	0.001
Add Link Chance	0.005
Remove Link Chance	0.0005
Selection Strategy	Truncation
Truncation Percent	0.3

Table 7: Evolution parameters for NEAT

Parameter	Value
Mutate Chance	0.1125
Mutate Type	Weight Perturb
Mutate Amount	1
NEAT Crossover Rate	0.5
Add Node Chance	0.00005
Add Link Chance	0.005
Remove Link Chance	0.00005
Selection Strategy	Truncation
Truncation Percent	0.3

Table 8: Evolution parameters for HyperNEAT

B Evolution

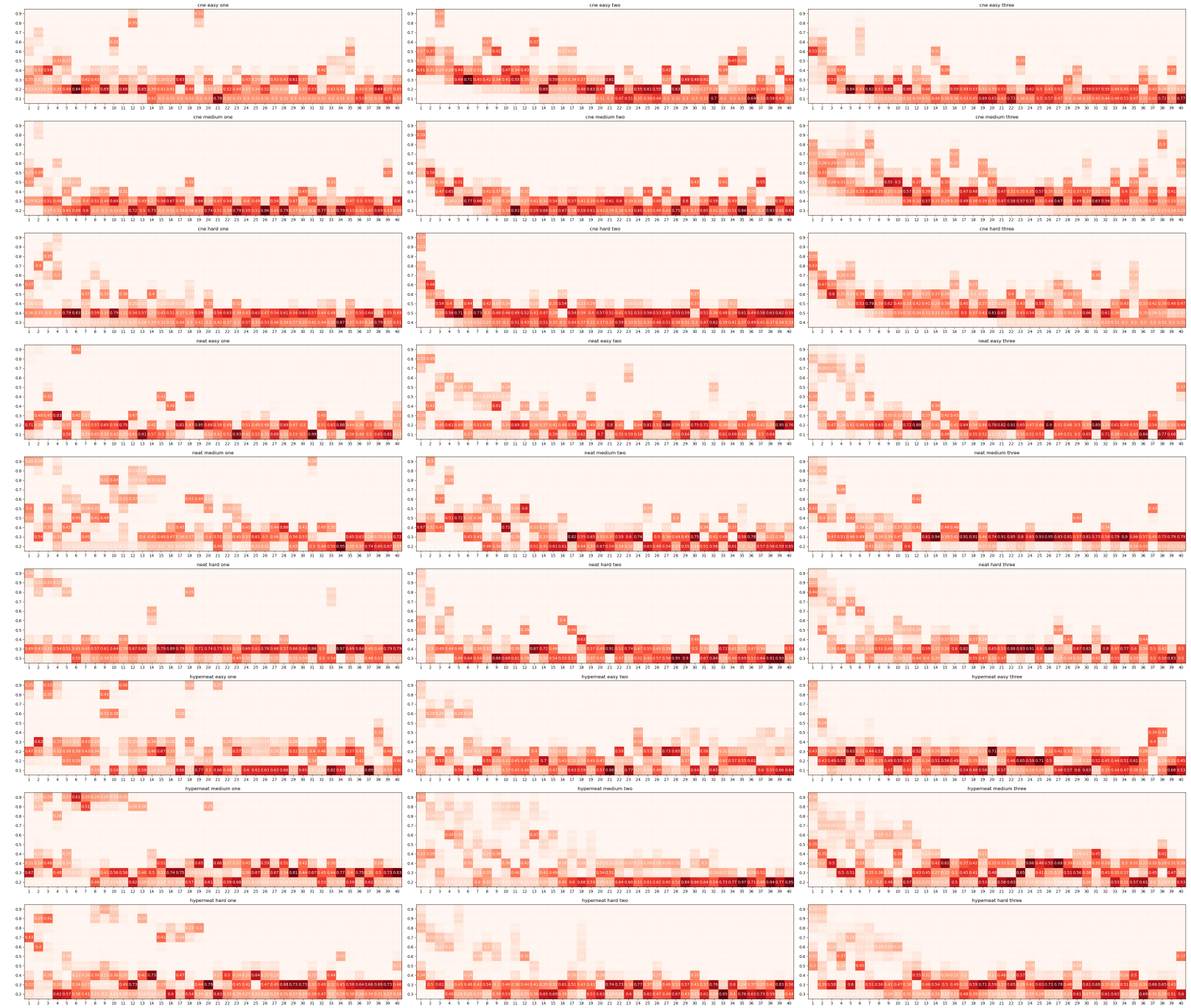

Figure 23: Exploration heat map for each controller during training

C Normality

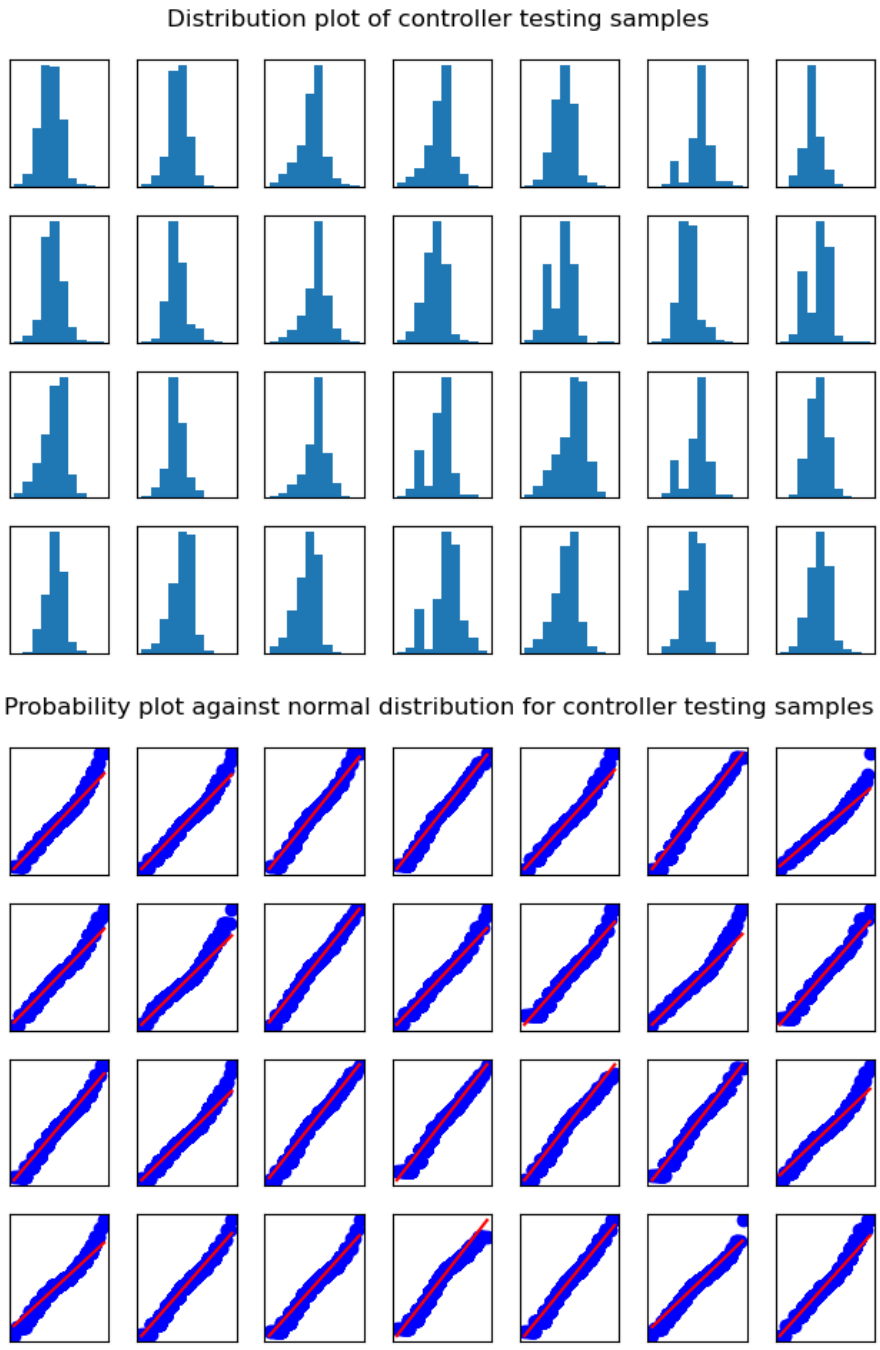

Figure 24: Distribution for testing results of each controller

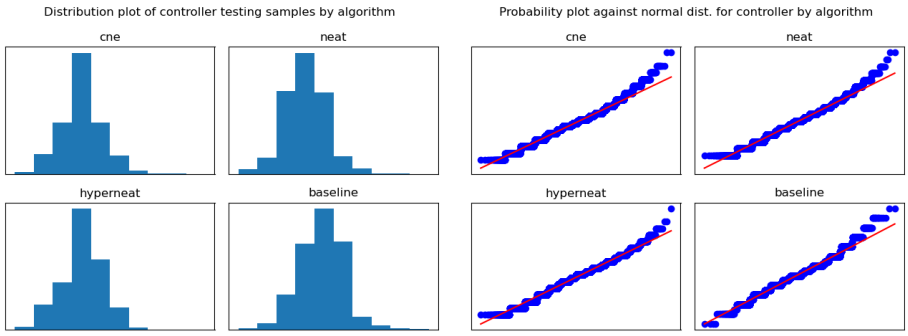

Figure 25: Distribution for testing results of each algorithm

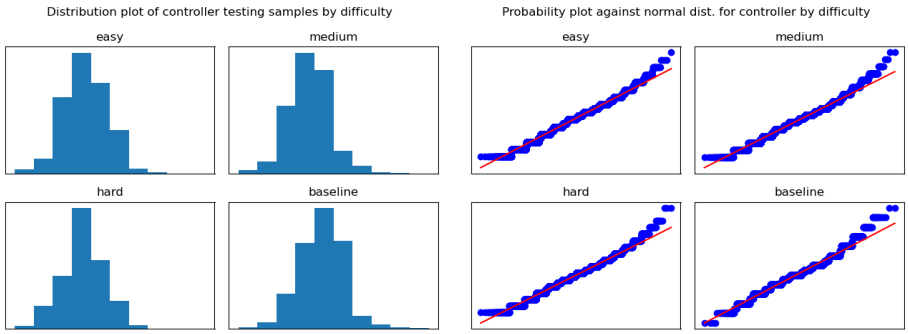

Figure 26: Distribution for testing results of each difficulty

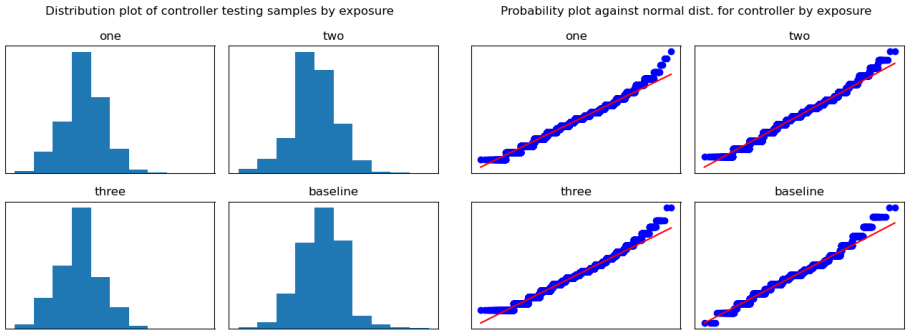

Figure 27: Distribution for testing results of each exposure

D Significance Testing

	CxEx1	CxEx2	CxEx3	CxMx1	CxMx2	CxMx3	CxHx1	CxHx2	CxHx3	NxEx1	NxEx2	NxEx3	NxMx1	NxMx2	NxMx3	NxHx1	NxHx2	NxHx3	HxEx1	HxEx2	HxEx3	HxMx1	HxMx2	HxMx3	HxHx1	HxHx2	HxHx3	BxBxb
CxEx1	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CxEx2	0.00	x	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00
CxEx3	0.00	0.01	x	0.00	0.04	0.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.80
CxMx1	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.64	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CxMx2	0.00	0.00	0.04	0.00	x	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.07
CxMx3	0.00	0.00	0.55	0.00	0.14	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.73
CxHx1	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CxHx2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.40	0.00	0.00	0.00
CxHx3	0.00	0.00	0.00	0.64	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NxEx1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00	0.91	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NxEx2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.71	0.00	0.00	0.00	0.00
NxEx3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NxMx1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NxMx2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NxMx3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.14	0.00	0.00	0.00	0.53	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NxHx1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.14	x	0.00	0.00	0.00	0.41	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.00
NxHx2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.91	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NxHx3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
HxEx1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
HxEx2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.53	0.41	0.00	0.00	0.00	x	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00
HxEx3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00	0.00
HxMx1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00	0.00
HxMx2	0.00	0.99	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00	0.00
HxMx3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.71	0.00	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.02	0.00	0.00	0.00	x	0.00	0.00	0.00	0.00
HxHx1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00	0.00
HxHx2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00	0.00
HxHx3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x	0.00
BxBxb	0.00	0.00	0.80	0.00	0.07	0.73	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	x

Table 9: T test results when comparing all controllers' performance against each other

	cne	neat	hyperneat	baseline
cne	x	0.00	0.00	0.00
neat	0.00	x	0.00	0.00
hyperneat	0.00	0.00	x	0.00
baseline	0.00	0.00	0.00	x

	easy	medium	hard	baseline
easy	x	0.00	0.00	0.76
medium	0.00	x	0.00	0.00
hard	0.00	0.00	x	0.00
baseline	0.76	0.00	0.00	x

	one	two	three	baseline
one	x	0.00	0.00	0.00
two	0.00	x	0.00	0.02
three	0.00	0.00	x	0.00
baseline	0.00	0.02	0.00	x

Table 10: T test results when comparing factors against each other

E Behavior

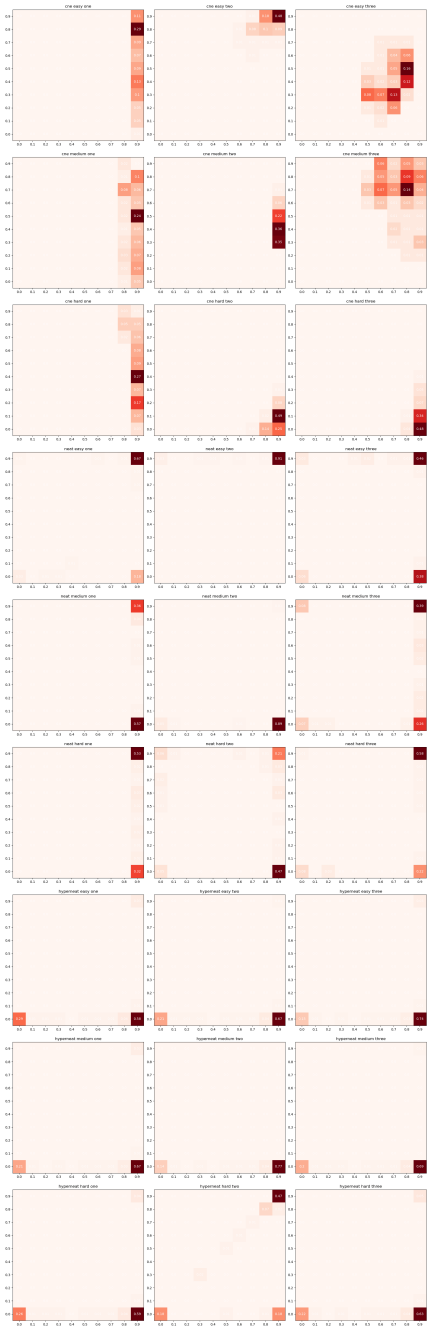

Figure 28: Network output heat map for each controller during testing

F Network Structure

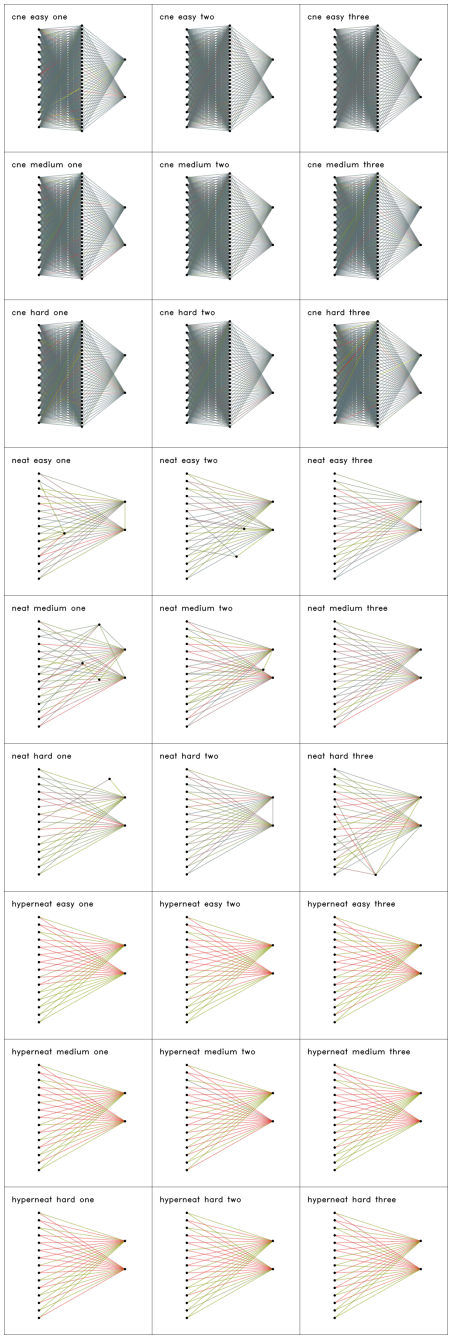

Figure 29: Structure of each controller's neural network