UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Department of Computer Science

Computer Science Honours Final Paper 2016

Title: An Implementation and Comparison of Centralised and Decentralised Approaches to Coordinated Driving

Author: Aashiq Parker

Project abbreviation: DS-DC

Supervisor: Geoff Nitschke

Category	**Min**	**Max**	**Chosen**
Requirement Analysis and Design	0	0	0
Theoretical Analysis	0	0	0
Experiment Design and Execution	0	20	20
System Development and Implementation	0	15	5
Results, Findings and Conclusion	10	20	20
Aim Formulation and Background Work	10	15	15
Quality of Paper Writing and Presentation	10		10
Quality of Deliverables	10		10
Overall General Project Evaluation	0	10	
Total marks			80

An Implementation and Comparison of Centralised and Decentralised Approaches to Coordinated Driving

Aashiq Parker Computer Science Department (University of Cape Town)
Rondebosch, 7700
Cape Town, South Africa
amp940703@gmail.com

ABSTRACT

This study presents an investigation into how well a decentralised approach for coordinated driving using neuro-evolution performs against a deterministic centralised approach. Furthermore, we attempt to establish whether a direct or indirect encoding scheme is more suitable for the decentralised approach. A custom simulator with an implementation of the centralised approach was built and used as a platform to execute the neuro-evolution experiments. Each approach was evaluated over ten tracks representative of realistic occurrences in road networks that are not usually tested in similar studies. The results indicate that while the centralised approach performs well over all tracks, the decentralised approach performs better on tracks with less common occurrences such as traffic circles. Furthermore, we determine that the indirect encoding scheme is less suited for the specific task due to the lack of regularity in the task domain.

Keywords

Machine learning; Neuro-Evolution; Autonomous Intersection Management; Self-driving cars; NEAT; HyperNEAT

1. INTRODUCTION

Autonomous vehicles are a popular area of study in machine learning that has seen great interest due to the increased funding by prominent companies. These companies, like Tesla, are embarking on an arms race in the hopes of successfully commercializing self-driving cars by 2020 [24]. This study investigates ways to automatically produce decentralised car control-systems for future autonomous vehicle applications by using machine learning and compares the performance against a centralised approach.

Using machine learning, there are two important tasks involved in the design of self-driving cars. Firstly, sensors must be placed in manner that maximises the information received from the environment but minimizes the energy consumption levels of the vehicle. Secondly, car agents must be able to use the information from these sensors in order to make a series of decisions that will ultimately navigate the car through its environment.

Neuro-evolution (NE) is a machine learning technique that combines two prominent biologically-inspired methods, that is, *artificial neural networks* (ANNs) and *evolutionary computing* (EC) [25] and is an unsupervised learning method - it does not depend on training data in order to learn the required behaviours. A typical NE application requires that a mapping is defined from each *genotype* (a potential solution stored in memory) to its *phenotype* (a potential solution that can be evaluated on the task). This mapping can either be *direct* (phenotypes are explicitly stored in memory as genotypes) or *indirect* (a mapping function is used to translate from genotype to phenotype). Two similar NE approaches that employ such encodings are the *Neuro-Evolution of Augementing Topologies* (NEAT - the direct encoding variant) [31] and the *Hypercube-based Neuro-Evolution of Augmenting Topologies* (HyperNEAT - the indirect encoding variant) [30]. Prior works have seen little application of NE as a method to automatically produce coordinated driving behvaiours for multiple cars [6, 7, 17]. Furthermore, the efficacy of using a direct encoding versus an indirect scheme for the task has received even less attention. This study focuses on the problem of coordination.

Currently, we use mechanisms such as traffic intersections in order to manage coordination between vehicles. However, these mechanisms were designed for human drivers and so do not exploit the full capabilities of autonomous vehicles. While certain works focus on optimizing traffic flows by investigating traffic light behaviours [28, 4, 19, 14], one approach that completely deprecates the use of traffic lights is *Autonomous Intersection Management* (AIM) [12]. AIM is a scalable technique for coordinating traffic at intersections without any traffic lights. AIM replaces traffic lights at intersections with *intersection managers*. These managers are responsible for determining which vehicles should be allowed through the intersection on a first-come first-serve (FCFS) basis. AIM assumes only the presence of autonomous vehicles and is a deterministic method - it does not make use of machine learning principles to develop solutions. Furthermore, AIM is a centralised approach - all vehicles need to communicate with the set of intersection managers in order to manage coordination.

1.1 Problem Statement and Research Goals

This study investigates the capability of NE producing ANN controllers that exhibit behaviours for managing coordination amongst self-driving cars. These controllers are responsible for getting cars through a road network whilst minimizing the time taken and collisions made. Each simulated vehicle is autonomous, that is, each vehicle is only capable of using the information at its disposal to inform its decisions and does not communicate with any other simulated vehicles. Furthermore, the controllers that emerge from the interactions between the simulated vehicles are decentralized - they do not interact with any third-party mechanisms. The research objectives are as follows:

1. The first research objective is to test how a decentralised approach for coordinated autonomous driving created using HyperNEAT performs against the centralised AIM approach, where the task performance of both approaches are evaluated on a range of roads.

2. The second research objective is to test how a decentralised approach for coordinated autonomous driving which employs indirect encoding (HyperNEAT) performs against the same decentralised approach using direct encoding (NEAT), where the task performance of both approaches are evaluated on a range of roads.

Here task performance is measured based on three factors, that is, speed, collisions made and car throughput. While controllers must seek to maximise the speed and throughput of vehicles, they must minimize collisions between vehicles.

Section 2 of this study provides a background for the approaches that were used in the experiments. Section 3 provides an overview of the methods and experimental setup. Finally, Section 4 presents the results and provides a discussion thereof and is followed by the conclusions.

2. BACKGROUND

In the following subsections we first provide an overview of AIM - a popular, centralised approach used as one of the comparative methods of this study. We then review NE and introduce two of its applications, NEAT and HyperNEAT, used for creating the decentralised approaches and each employing a different encoding scheme.

2.1 Autonomous Intersection Management

Autonomous Intersection Management (AIM) [12] is a deterministic approach that efficiently coordinates vehicles without the need for traffic lights by using a centralised reservation-based approach. AIM employs a set of *intersection managers*, one at each intersection. Vehicles must then communicate with these intersection managers via requests before entering an intersection. Intersection managers are responsible for processing these requests and subsequently granting or denying the car permission to enter by simulating the car's path through the intersection and checking for collisions. This process is described in figure 2. As the tracks used in AIM are discretised into cells, the allocation of a path by the intersection manager is simplified as individual cells can reserved at any time step.

Because the intersection manager gathers requests over time, different policies can be employed in deciding which vehicles to let through first. The first versions of AIM use a FCFS reservation policy in which vehicles that transmit requests first (and hence arrive at the intersection first) are allowed to enter, provided that there will be no collisions with other vehicles that have reserved a path. This way, vehicles receive a response from the intersection manager immediately once the request has been processed. Stone et al. [1] later introduced a batch-processing policy in which requests are first stored into a buffer and then later processed in batches. This way, the intersection manager can prevent starvation by taking factors such as the relative waiting times into account. For the purposes of this project, only FCFS-AIM was implemented and tested due to time constraints and for the sake of simplicity.

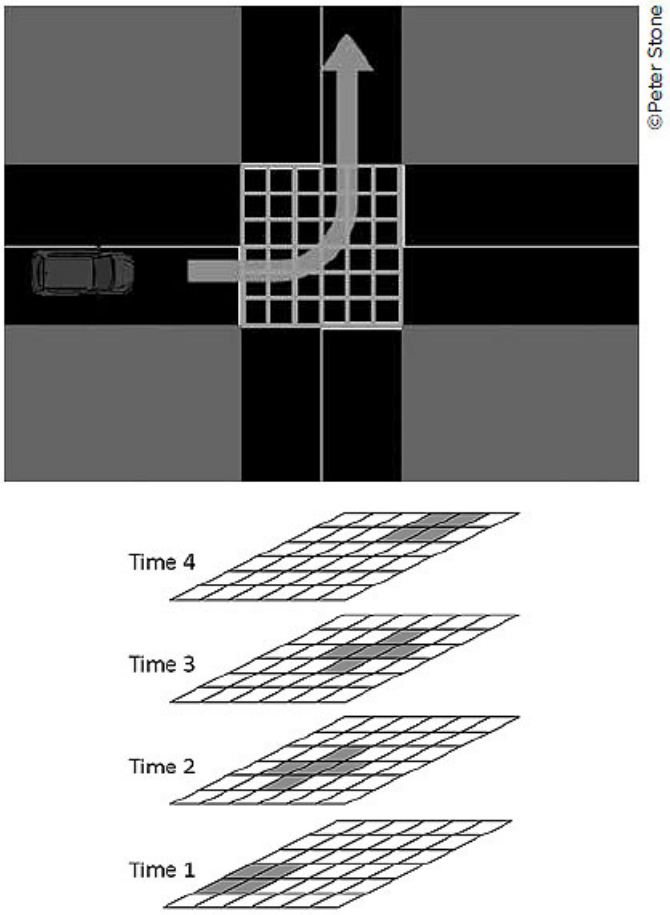

Figure 2: AIM simulation of a car's path through an intersection [33]: An intersection manager uses the car's expected position and time to create a list of positions (cells) and times at which the car might potentially navigate through the intersection. The cells and times will be reserved for the vehicle if there are no collisions with previously reserved cell-time tuples.

2.2 Neuro-Evolution

Neuro-evolution (NE) is a machine learning technique that uses Evolutionary Computing (EC) to develop Artificial Neural Networks (ANNs) [25]. NE provides an alternative way of training ANNs when compared to traditional machine learning and can prove beneficial when certain parameters about the network is not known, making it a much more general approach [16].

Artificial neural networks (ANNs) are layers of interconnected information-processing nodes (neurons) that are inspired by the human nervous system [2]. These kinds of networks are popular because of a number of characteristics which include, but are not limited to, massive parallelism capability, fault tolerance, generalisation ability and are capable of approximating any continuous function [20]. Hence, ANNs are well suited as agent controllers that must operate in realistic environments due to their ability to deal with noisy and incomplete information.

Evolutionary computing (EC) is a subarea of machine learning that bases its foundation on the Darwinian princi-

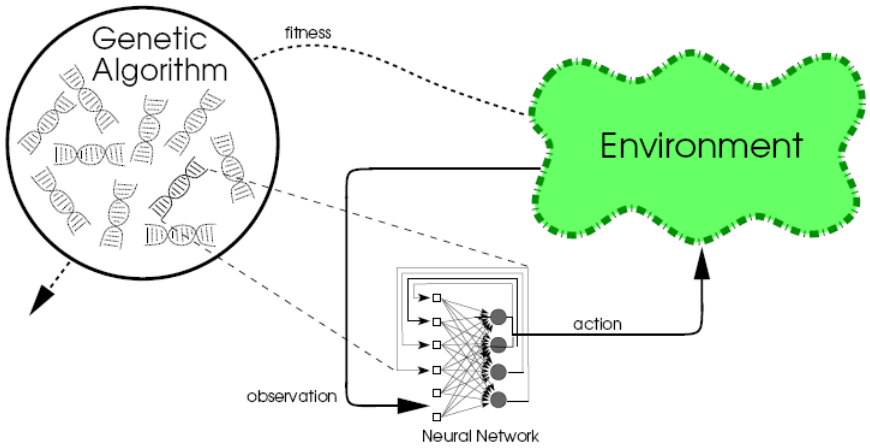

Figure 1: General structure of a NE method [18]. A population of ANNs, encoded as chromosomes, are evolved for a specific task using a type of evolutionary algorithm called a *genetic algorithm*. Each ANN receives input to its network using information from the environment and computes some output. The ANN is evaluated and assigned a fitness value using a fitness function that measures how well it performs on the task. ANNs that perform well are allowed to mate and propagate their genetic material into future generations.

ples of natural selection and adaptation [13]. The underlying model of a problem in evolutionary computing is defined by a population of individuals, an environment in which the individuals exist and a measure of the quality of the individuals. EC adopts a generational approach to produce increasingly better and novel solutions using genetic operators such as *mutations* and *recombination*.

NE is thus capable of combining the advantages of both components by using EC to evolve ANNs. Using NE, evolution of an ANN is typically in one of the following ways: 1) evolving the connection weights between neurons in an ANN; 2) evolving the learning rules that neurons in an ANN use; 3) evolving the topology of an ANN; or 4) evolving a combination of the weights, topology and learning rules [34]. As with other algorithms that fall under EC, NE evolves population of individuals (the individuals now being ANNs) using a measure of fitness to decide which individuals should be able to propagate their genes through a number of generations. Figure 1 details the general structure of an NE algorithm.

As with all applications of EC, individuals in NE are represented as genotypes which in turn encode phenotypes that are evaluated by some fitness function. Applications of NE fall into two categories based on the type of encoding they use, which is either *direct encoding* or *indirection encoding*. A third hybrid encoding approach also exists [9] which attempts to combine the benefits of both types of encoding, however not much research has gone into evaluating this approach. The two algorithms used in the experiments presented in this study, each employing one of the encoding variants, are described here.

2.2.1 Neuro-Evolution of Augmenting Topologies

Neuro-evolution of Augmenting Topologies (NEAT) [31] is an NE method that makes searching for solutions in high dimensional space feasible by evolving ANNs. NEAT employs a direct encoding approach, that is, it explicitly stores all phenotypic information (nodes, connections between nodes and weights) in its genotype. NEAT uses three fundamental

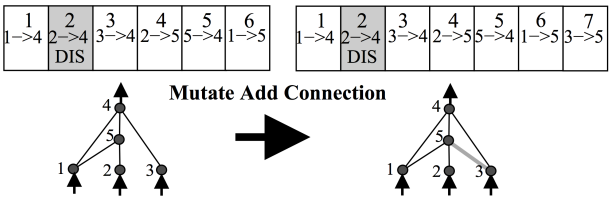

Figure 3: NEAT mutation operator [31]: A structural mutation adds an extra connection to the ANN. The genotype representation for the connections between nodes is represented as a string. Each part of the string represents a connection between two nodes and whether the connection is enabled.

principles that ensures the efficient evolution of the topology and weights of complex networks , that is: 1) ensuring that the initial population of individuals have a minimal configuration; 2) protecting new innovations that arise during the evolutionary process; and 3) tracking which parts of networks are suitable for exchange during crossover. Using such an approach mitigates many of the problems present in general NE [11, 31, 30]. Figure 3 displays how how a mutation might affect a genotype in NEAT.

NEAT is able to evolve ANNs that perform well in control and decision-learning problems [31, 11]. NEAT initalises individuals in a population with a minimal configuration and new structural components are only added when necessary. This is contrasted against starting the search in an intractably large space or with individuals that are initialised with random topologies. NEAT also employs speciation in order to protect new individuals that arise during the evolutionary process by dividing the population into niches. This way, new individuals are allowed the necessary time to develop features that could prove to be beneficial before competing with older individuals. Furthermore, as ANNs with different topologies can recombine, NEAT uses a global counter to historically mark genes as they appear. This way,

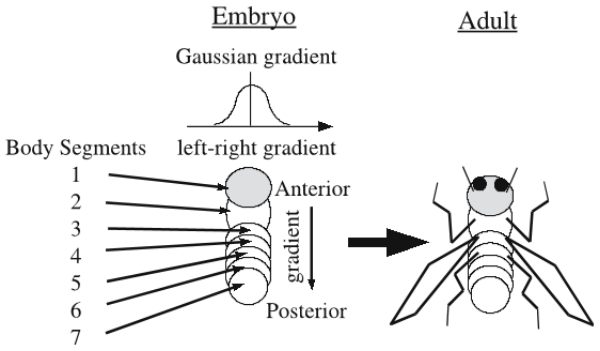

Figure 4: Simplified view of an insect's body plan [29]: Gradient's along the anterior-posterior and left-right axes are used to define the overall body plan of the insect early in development. The anterior-posterior axis provides a coordinate frame in which discrete segments of the insect's body can grow. Furthermore, a symmetric coordinate frame is used to establish bilateral symmetry along the left-right axis.

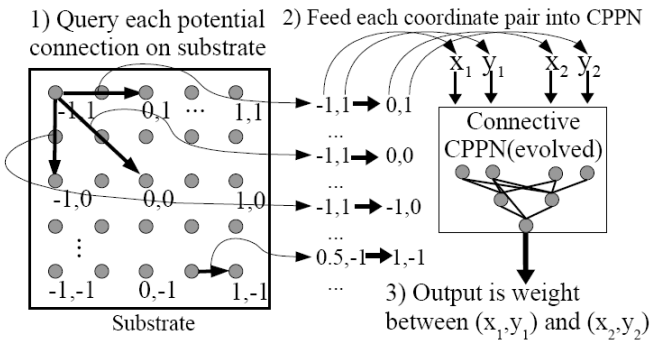

Figure 5: Hypercube-based Geometric Connectivity Pattern Interpretation [30]: The layout of nodes in space, called the substrate, is positioned and assigned coordinates such that the centre node is at the origin. 1) Every node is queried against every other node by 2) feeding the two nodes as input into the CPPN. 3) The CPPN outputs the weight between the two inputted nodes.

compatible parts of ANNs with differing topologies can be determined during recombination to result in meaningful offspring.

NEAT has been applied to a wide range of applications related to autonomous driving. These applications include evolving vehicle controllers for racing games and simulations [6, 5]. Furthermore, NEAT has also been used to evolve controllers for complex group behaviours [27], producing favourable results.

2.2.2 *Hypercube-Based Neuro-Evolution of Augmenting Topologies*

In order to model complex and sophisticated phenotypes in a space efficient manner, we need to be able to map from a space of few dimensions (genotype) to a second space that contains many (phenotype). *Hypercube-Based Neuro-Evolution of Augmenting Topologies* (HyperNEAT) [30] is an extension of NEAT that does exactly this by employing an indirect encoding scheme. It does so by evolving a population of *connective compositional pattern-producing networks* (connective CPPNs). These CPPNs are in turn responsible for encoding ANNs that are evaluated on the specific task.

CPPNs are abstractions of natural development. They are capable of producing complex patterns and regularities by composing simple functions. Hence, they are analogous to ANNs that are allowed to have different activation functions at each node. These regularities, such as symmetry and repetition, are similar to those prevalent in nature. The idea is that each function in the CPPN acts as a *coordinate frame* in which other functions can reside and are abstractions of developmental events, such as establishing bilateral symmetry by using symmetric function. This is displayed in figure 4 where by using a symmetric Gaussian function, the insect's discrete body segments are able to grow wings and legs on both sides while only having to define them once. This is contrasted against traditional neuro-evolution methods where the same functions are developed isolation instead of being reused. By evolving CPPNs that decode ANNs, HyperNEAT is able to efficiently produce ANNs with connectivity patterns much like the neural connectivity patterns of the brain. Furthermore, because of its encoding approach, HyperNEAT is capable of producing much larger ANNs than previous machine learning approaches are capable of doing [30].

Using HyperNEAT, an initial *substrate configuration* needs to be specified first. The substrate is a layout of the nodes of an ANN in space, essentially assigning a position to each node. This way, the substrate can query a connective CPPN using two nodes as input to determine the weight of the connection that joins the two nodes. This approach essentially allows HyperNEAT to *see* the geometry of the problem and produce connectivity patterns that are a function of the problem geometry [29]. Consider, as an example, learning to play the game tic-tac-toe. Using the 9 cells of the grid as input to the ANN, most neuro-evolution applications are completely blind to the geometry of the problem. These inputs would then just be an array of values that may or may not be related and the network then has to infer over hundreds of generations that they indeed are. HyperNEAT, however, is able to see this geometry which can significantly increase learning performance and reduce convergence times [30]. Figure 5 shows an example of how a substrate is queried by a connective CPPN to form a ANN. In this example, each node in the substrate is queried against every other node to determine its weight. This is done by providing the positions of the two nodes as input to the connective CPPN. The dark lines represent some of the nodes that are queried. HyperNEAT is thus able to exploit the geometry of the problem as the connectivity pattern produced is a function of the geometry of the substrate.

As HyperNEAT is an extension of NEAT, the constructs used in the algorithm is the same as the constructs employed in NEAT. Furthermore, the constraints that apply to the ANNs produced through NEAT also applies to those created through HyperNEAT. While HyperNEAT has limited application to autonomous driving, it has been applied to many related applications. These include evolving the coordination behaviour of robots [8] and this has seen favourable results. Furthermore, HyperNEAT has also been applied to develop group behaviours for evolutionary robotics [3].

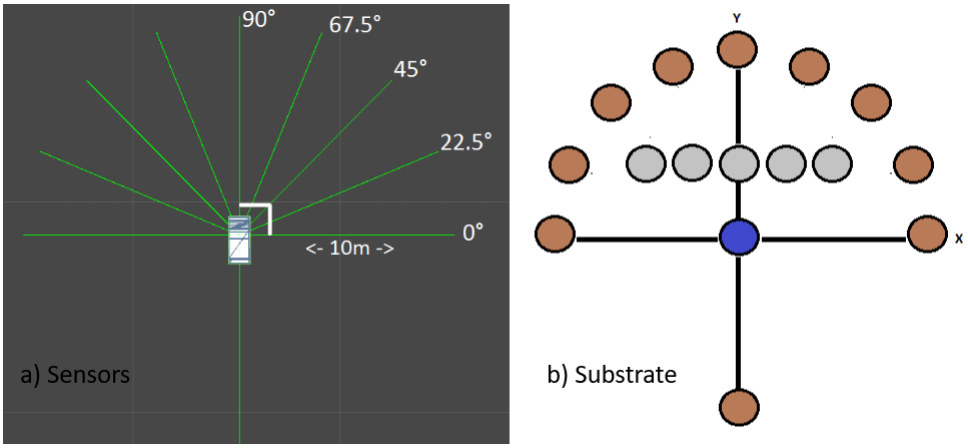

Figure 6: Sensory and substrate configurations: Configurations for a) The layout of sensors that each car agent uses. Each car is equipped with nine range finding sensors spanning 180 degrees from the left of the car to the right and one range finding sensor to detect objects behind the car; b) The substrate setup for HyperNEAT that gemoetrically mimics the positions of the car sensors. The nodes coloured brown are the input nodes, grey are the hidden nodes and blue the output node.

3. METHODS

The following subsections present how the two approaches to coordinated driving behaviour were implemented for the experiment, that is, the centralised AIM and the decentralised HyperNEAT approaches. Furthermore, we provide an overview of the experiment and how our approaches were evaluated.

3.1 AIM

Our implementation of AIM [12] uses the reservation-based approach by placing an intersection manager at each intersection. Each autonomous vehicle communicates with an intersection manager as it arrives at the respective intersection. This is contrasted against the original implementation of AIM, where cars communicate with intersection managers as they enter a new a lane, however this is not necessary as only the FCFS protocol was implemented. When the vehicles reach an intersection, they simulate a path and produce a list of time-position tuples. Each position is a 2D vector instead of a cell position as custom tracks that cannot easily be discretised into cells were used. This information is then bundled into a request that gets sent to the intersection manager in attempt to reserve a path through the intersection. The intersection manager stores a list of all reserved position-time tuples and uses this to check incoming requests for collisions. If the intersection manager does grant the car access, it will store these tuples in memory and use this information for future requests. Vehicles that do not have there request approved then have to wait before retransmitting a request and will ultimately stop at the intersection only if it does not get its request approved.

Our implementation employs the FCFS policy, that is, vehicles that transmit requests first will be allowed to enter the intersection first, provided there are no collisions. Although the original version of AIM allows for cars of various sizes, only a single car size was used for simplicity purposes. Furthermore, a set of heuristics not included in AIM were added to prevent cases where lanes would overflow. These are availabe online.

3.2 HyperNEAT

An adaptation of the HyperNEAT algorithm for C#, called SharpNEAT[1], was used to execute our experiments within our simulator. The HyperNEAT particulars are discussed below.

3.2.1 *Sensory Configration*

For the experiment, each car agent is equipped with ten range-finding sensors. The sensors are arranged such that nine of them span 180 degrees at equal intervals along the car's left-right axis. The last sensor points in the car's backwards direction. The sensory setup is depicted in figure 6. Each sensor is responsible for providing the distance that the closest obstacle is to the car along the respective sensor's direction.

3.2.2 *Substrate Configuration*

Each node in the ANN is placed on a 2D substrate as displayed in figure 6. The nodes are arranged in manner that closely mimics the geometry of the problem. The ten input nodes (coloured brown) are placed at the end-points (furthest distance at which an obstacle can be detected) of the respective range-finding sensors. The nodes are arranged symmetrically about the y-axis and the output node is placed at the origin of the substrate (coloured blue). Lastly, five hidden nodes (coloured grey) are placed on a horizontal line between the input and output nodes.

Using the ten range-finding sensors, the ANNs receive ten inputs corresponding to the distance of the closest obstacle along the direction of each sensor. These values are inverted and normalised between 0 and 1. A value close to 1 means that the obstacle is very close, a value close to 0 means that the obstacle is far away and a value of 0 means that no obstacle has been detected. Furthermore, the single output node is a weight that controls the vehicle's speed. A value of 0 means the car is not moving aand a value of 1 means the car is travelling at full speed.

[1]http://sharpneat.sourceforge.net/- Date Accessed: 2016-11-01

NE Parameters	
Simulation Runs	20
Number of Trials	3
Trial Duration	40 seconds
Population Size	150
Generations	150
Species	5
Selection Operator	Roulette Wheel
Activation Scheme	Acyclic
AIM Parameters	
Protocol	FCFS
Trial Duration	90 seconds
Vehicle Speed	8
Request Cooldown	1.5 seconds

Figure 7: Experiment Parameters. The list of parameters used for the AIM and NE experiments.

3.3 Experimental Setup

Our simulator was built using the Unity[2] game engine. The simulator was built in order to execute the HyperNEAT experiment as well as to evaluate the resulting HyperNEAT controllers and the implementation of AIM. The purpose of these experiments were to to test the research objectives as outlined in Section 1.1 and hence the experiments were constructed accordingly. These particulars are discussed here.

3.3.1 Tracks

A total of 10 tracks were created and imported into the simulator using a free 3D modelling software called Blender[3]. Our tracks differ from conventional tracks used for similar studies in that they are inspired by actual road networks around the University of Cape Town. Each track is defined by a set of nodes that are responsible for demarcating the different types of paths, for example, an intersection or a usual driving road. This is displayed in figure 8 (the full list of tracks is available online). The paths between nodes are defined using Bezier curves. Using these curves, a set of way points can be calculated which the car needs to follow. Thus, a car can be assigned a random path from a start node to and end node and the way points can be calculated using the intermediate Bezier curves. This way, each car has complete information with regards to the route it has to take as the simulated vehicles must learn to coordinate with each other, not how to drive.

3.3.2 AIM Experiment

Each experiment tests AIM's ability to produce coordinate driving behaviour for a particular track, where the experiment constitutes 20 runs of AIM each lasting 90 seconds. The results recorded are the throughput and speed of vehicles, the number of collisions and the total idle time. Figure 7 displays the AIM simulation parameters used.

3.3.3 HyperNEAT Experiment

Each experiment uses HyperNEAT to evolve controllers that exhibit coordinated driving behaviours for a particular track, where the experiment constitutes 20 runs of 150 generations with a population size of 150. Furthermore, each individual is evaluated over three trials lasting 40 seconds each. Each inidividual is evaluated according to the following fitness function:

$$f = (S * 2000) + (D * 1000) - (C * 250) - (B * 100) \quad (1)$$

where S is a value between 0 and 1 and represents the average speed across all the vehicles in the trial, D is a value between 0 and 1 and represents the average minimum distance to an obstacle across all vehicles, and C and B are integers representing the number of collisions and cases where stationary cars prevented cars from being spawned respectively. As the average speed (S) is multiplied by the largest weight, controllers get the most fitness by constantly moving cars and hence getting cars to their destination. The distance (D) variable encourages cars to keep a driving distance between them, making collisions less likely. Furthermore, controllers are penalised whenever a car stalls and prevents other cars from entering a lane. They are also penalised for any collisions - cars that collide are removed from the simulation. Figure 7 displays the NE parameters used. The full set of parameters is available online.

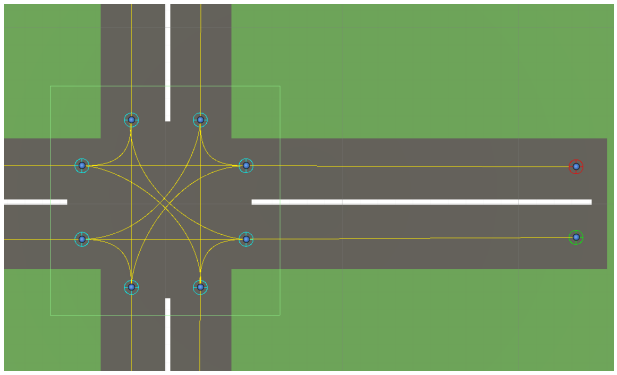

Figure 8: Paths and Nodes Each path is defined a set of nodes which in turn defines a set of Bezier Curves used to calculate way points.

4. RESULTS AND DISCUSSIONS

The results indicate that while centralised FCFS-AIM generally performs well across all tracks, a decentralised approach utilising very limited information is capable of producing better behaviours for some of the tracks. Specifically, NEAT outperforms FCFS-AIM on certain tracks however HyperNEAT performs the worst across all tracks as it is incapable of producing driving behaviours without any collisions. This result is supported by a statistically significant difference ($p < 0.05$, two-tailed t-test [15]) between the decentralised and centralised approaches over all tracks.

Figure 9 displays the percentage throughput of cars averaged over 20 runs for each approach and track, with error bars indicating one standard deviation about the mean. Figure 10 displays the average collisions, averaged over 20 trials. AIM results for collisions are not included in this figure as AIM is designed such that no collisions ever occur. Furthermore, figures 11 and 12 display the average vehicle speed

[2]https://unity3d.com/ - Date Accessed: 2016-11-01
[3]https://www.blender.org/ - Date Accessed: 2016-11-01

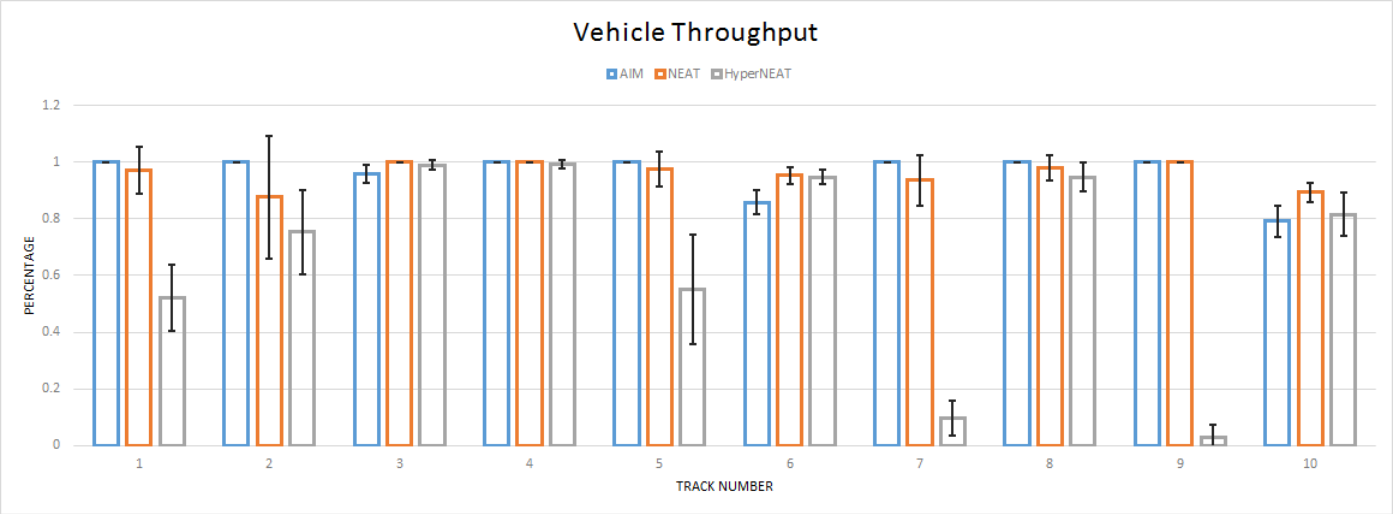

Figure 9: Vehicle throughput per track. The chart displays the percentage of cars that successfully arrived at their destinations per track, averaged over 20 trials.

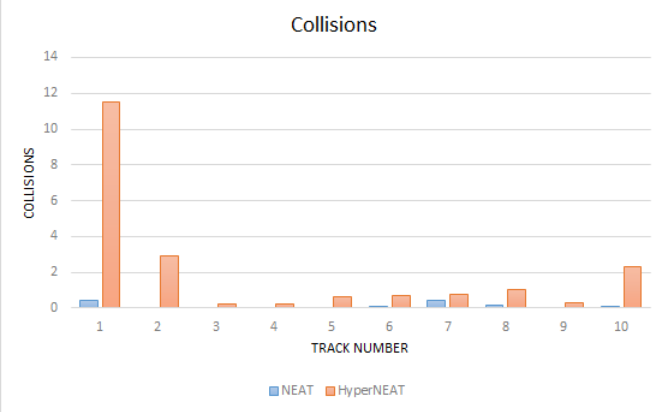

Figure 10: Average collisions per track. The chart displays the average number of collisions that occured per track, averaged over 20 trials. Cars that collide are removed from the trial.

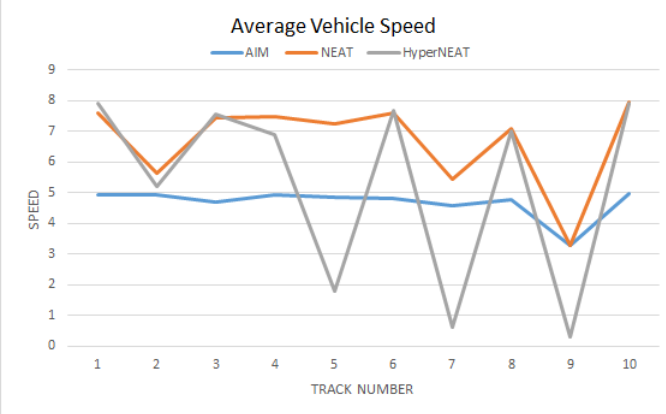

Figure 11: Average vehicle speed per track. The chart displays the average vehicle speed per track, averaged over 20 trials

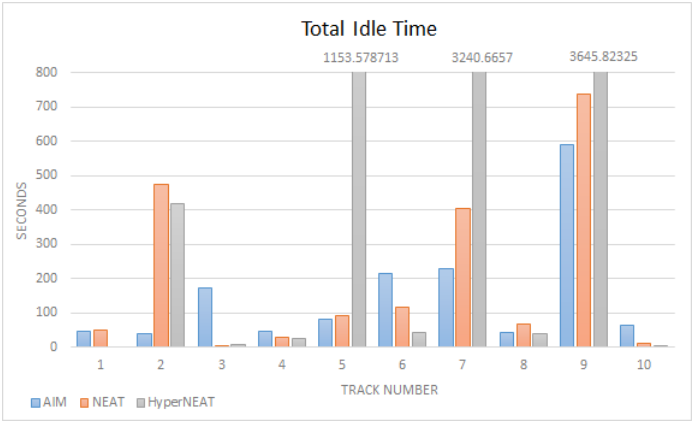

Figure 12: Total idle time per track. The chart displays the total idle time of all vehicles per track, averaged over 20 trials.

and total idle time respectively, averaged over 20 trials. The results for NEAT are included for comparison. Information about the precise NEAT setup and experiments are available online as part of a similar study.

AIM generally produces good results, being able to obtain a 100% throughput rate across seven of the tracks. However, AIM does not produce optimal results for tracks 3 (traffic-circle), 4 (double-lane merge) and 9 (one-way merge). When viewing the tracks that AIM performs well on, we see that they are mostly similar. For instance, the only difference between track 1 and track 2 is the number of intersections. Similarly, track 5 differs from track 2 in that it has more lanes. These are contrasted against 3,4 and 9 - they are less common occurrences and are very seldomly tested in related works. Furthermore, figure 11 indicates that average vehicle speed in AIM is not optimal. This is because vehicles in AIM do not enter an intersection until a path with no collisions can be guaranteed. This also impacts the idle time of vehicles as can be seen for track 3 in figure 12

where vehicles end up waiting long periods of time at intersections. These results indicate that AIM performs well on common and uncommon roads, however its performance is not optimal on uncommon roads when peculiarities such as one-ways or traffic circles are introduced. Previous studies only test AIM on common roads similar to track 2, where the number of lanes and intersections vary [12, 11, 1].

Furthermore, the results indicate that NEAT is capable of producing good coordinated driving behaviours for the uncommon roads. NEAT also performs well for many of the common tracks however collisions do still occur in most cases. This is because while controllers must minimize collisions according to the fitness function, it is not an absolute requirement - controllers can achieve high fitnesses while still having a few collisions. The amount of collisions, however, are very low. Figure 10 displays that NEAT controllers always average zero or less than one collision per track. The results also show that the vehicles using NEAT controllers always drive at an equal or greater speed than those of AIM and HyperNEAT.

HyperNEAT was incapable of producing any good coordinated driving behaviours - all controllers over all tracks exhibited collisions. Furthermore, HyperNEAT produced lower throughput rates and higher idle times for most tracks, particularly for tracks 5, 7, 9. When correlating the speed, idle time and throughput of these three tracks, it is revealed the HyperNEAT adopts a strategy whereby vehicles stall upon an imminent collision. This creates a ripple effect and all vehicles end up stalling until the end of the trial, producing a low throughput rate, low speeds and high idle times. Figure 13 displays the resulting ANN (encoded by the evolved CPPN as described in Section 2.2.2) of HyperNEAT for track 5 (the full list of networks produced is available online). Here, HyperNEAT attempts to establish meaningful regularities in the form of connectivity patterns. However, the results indicate that the regularities discovered are not well suited for the task and instead produces an ANN that performs poorly because of its high complexity.

A *fractured problem* can be defined as a problem in which the output varies significantly between agent states [22]. We hypothesise that the coordinated driving task presented in this study is indicative of a fractured problem because of the experimental setup. More specifically, the decentralised HyperNEAT approach does not make use of path information to make decisions, hence creating a lack of regularity within the task domain. Furthermore, a *regular problem* can be defined as a problem in regularities exist, that is, patterns or motifs that are repeated in phenotypes [10]. HyperNEAT performs well on regular problems because of its ability to discover and exploit these regularities [30]. This is supplemented by its ability to 'see' the problem geometry as it essentially tries to establish a meaningful relationship between the problem geometry and these regularities. The opposite, however, is also true - HyperNEAT and NEAT perform badly on fractured problems and HyperNEAT performs particularly poorly on irregular problems [23]. These problems usually involve strategic decision-making [21] and an example is the classic concentric spirals machine learning benchmark [26].

In the experiment, HyperNEAT is used to evolve controllers that only use their sensory information as input to the ANNs, that is, ten values representing the distance to obstacles around the car and no path information. This is done as cars already have complete information with regards to their path. However, this introduces an irregularity - HyperNEAT is capable of seeing the geometry but it is incapable of establishing or discovering motifs and patterns. For instance, one motif might be to say that "if a left-side sensor detects an obstacle at a certain distance away but the vehicle is proceeding right, then the vehicle does not need to stop as a collision won't occur." HyperNEAT could then repeat this motif for the right sensors without having to redefine it. However, as the controller does use any path information as input, it has no sense of direction and is incapable of establishing these regularities and motifs and linking it to the problem geometry.

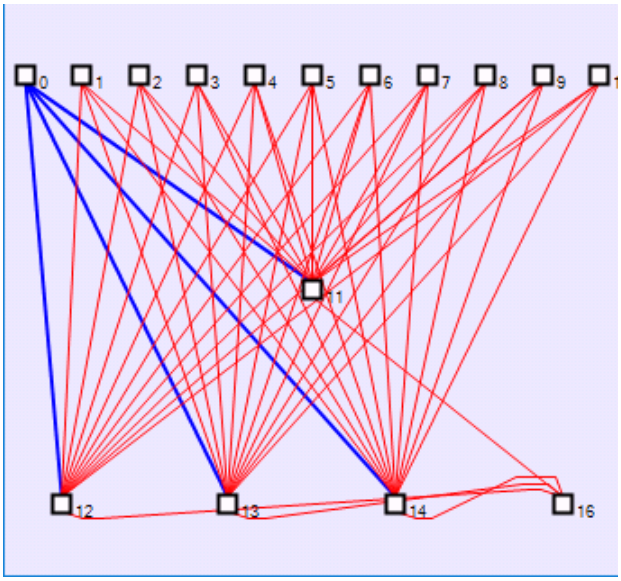

Figure 13: Resulting encoded ANN for track 5. The first ten nodes are input nodes, node 11 (centre) is the output node and the rest are hidden nodes. Blue lines are positive weights and red lines are negative weights. The line thickness represents connection strength.

These irregularities could provide a possible explanation as to why the best strategy evolved by HyperNEAT for tracks 3 (traffic circle), 7 (one-way merge) and 9 (double-lane merge) is to stall. The only cases where HyperNEAT can definitely tell that a collision is imminent is when an obstacle is close to its front sensor. As it cannot establish any other motifs that might support other strategies, the cars choose to stall instead to prevent the penalties associated with collisions. Results from a study that investigates the factors that cause HyperNEAT to fail [32] suggest that irregularities is the worst form of fracture, and hence that HyperNEAT performance degrades for such problems.

NEAT, on the other hand, still performs generally well even though the task is indicative of a fractured problem. This is because NEAT does depend on discovering regularities and patterns in the problem domain [31]. It instead learns to correlate specific input sensory values with specific output values, as does most direct encoding NE methods. Thus, it is capable of avoiding collisions by learning the specific cases in which obstacles are too close and subsequently result in a collision. These cases are easier to learn for specific tracks because of the sensory interactions. For

instance, when driving through a traffic circle in track 3, about two sensors will activate as there can only be a car in front and behind the vehicle. This is contrasted against a track with more lanes and intersections such as tracks 5 and 6. Here, there more lanes leading into an intersection and hence there are usually more cars inside the intersections. As a result, more sensors will activate and make it more difficult for NEAT to correlate these values with specific outputs.

Overall, the results do not suggest that a decentralised approach is ill-suited for coordinated driving behaviour. To the contrary, the results show how a decentralised approach using NEAT is capable of producing solutions that outperform the centralised FCFS-AIM approach for certain tracks, even though the approach uses very limited information and the specific task is fractured. HyperNEAT, however, is incapable of producing the expected behaviours and does not outperform AIM in any of tracks due to the lack of regularity. It is important to make the distinction here that the coordinated driving problem itself is not inherently a fractured problem [23], but that the specific setup used for this study resulted in the task domain being fractured. Furthermore, The results indicate that while AIM is a suitable and efficacious approach for common roads, a decentralised and non-deterministic approach using NEAT that is specific to the track is more suited for uncommon roads with peculiarities such as traffic circles. Finally, the results indicate that for the task presented in the study, an indirect encoding scheme is less suited than a direct encoding scheme.

5. CONCLUSIONS

The first purpose of this study was to investigate how a decentralised approach using the NE method HyperNEAT would perform against the centralised AIM approach for coordinated driving. Secondly, we wanted to establish which of two encoding schemes, direct or indirect, is more suitable for the decentralised approach to coordinated driving. A custom simulator was built and each approach was tested over a range of tracks. The results indicate that HyperNEAT performed poorly and was incapable of outperforming AIM or NEAT due to the lack of regularity in the specific task domain. The results also indicate that while AIM performs generally well over all tracks, NEAT produces better results for uncommon tracks with less common occurrences such one-way merges and traffic circles.

Furthermore, the results reiterate that the representation of the problem and how the task domain is setup for NE methods greatly affects the solutions that are produced. While HyperNEAT was not capable of producing the desired behaviours, the results still encourage the use of a decentralised NE approach to coordinated driving. Future work would entail testing the decentralised approach with different experimental setups. For instance, including path information as input to the ANNs to encourage regularity for HyperNEAT. Furthermore, the efficacy of a hybrid encoding scheme [9] for the task can be investigated.

6. REFERENCES

[1] T.-C. Au, N. Shahidi, and P. Stone. Enforcing liveness in autonomous traffic management. In *AAAI-11*, 2011.

[2] I. Basheer and M. Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43(1):3–31, 2000.

[3] J. Bongard. Evolutionary robotics. *Communications of the ACM*, 56(8):74–83, 2013.

[4] E. Brockfeld, R. Barlovic, A. Schadschneider, and M. Schreckenberg. Optimizing traffic lights in a cellular automaton model for city traffic. *Physical Review E*, 64(5):056132, 2001.

[5] M. Butz and T. Lonneker. Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 317–324. IEEE, 2009.

[6] L. Cardamone, D. Loiacono, and P. L. Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1179–1186. ACM, 2009.

[7] L. Cardamone, D. Loiacono, and P. L. Lanzi. On-line neuroevolution applied to the open racing car simulator. In *2009 IEEE Congress on Evolutionary Computation*, pages 2622–2629. IEEE, 2009.

[8] J. Clune, B. Beckmann, C. Ofria, and R. Pennock. Evolving coordinated quadruped gaits with the hyperneat generative encoding. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2764–2771. IEEE, 2009.

[9] J. Clune, B. Beckmann, R. Pennock, and C. Ofria. Hybrid: A hybridization of indirect and direct encodings for evolutionary computation. In *Advances in Artificial Life. Darwin Meets von Neumann*, pages 134–141. Springer, 2009.

[10] J. Clune, C. Ofria, and R. Pennock. How a generative encoding fares as problem-regularity decreases. In *International Conference on Parallel Problem Solving from Nature*, pages 358–367. Springer, 2008.

[11] D. D'Ambrosio and K. Stanley. Generative encoding for multiagent learning. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 819–826. ACM, 2008.

[12] K. M. Dresner. *Autonomous Intersection Management*. PhD thesis, The University of Texas at Austin, Austin, TX 78712, 2009.

[13] A. Eiben and J. Smith. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[14] M. Ferreira, R. Fernandes, H. Conceição, W. Viriyasitavat, and O. K. Tonguz. Self-organized traffic control. In *Proceedings of the seventh ACM international workshop on VehiculAr InterNETworking*, pages 85–90. ACM, 2010.

[15] B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes*. Cambridge University Press, Cambridge, UK, 1986.

[16] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[17] A. Gerdelan. Architecture design for self-training intelligent vehicle-driving agents: paradigms and tools. Technical report, Technical Report CSTN-088, Institute of Information and Mathematical Sciences, Massey University, North Shore 102-904, Auckland, New Zealand (April 2009), 2009.

[18] F. Gomez. *Robust non-linear control through neuroevolution*. PhD thesis, The University of Texas

at Austin, Austin, TX 78712, 8 2003.

[19] V. Gradinescu, C. Gorgorin, R. Diaconescu, V. Cristea, and L. Iftode. Adaptive traffic lights using car-to-car communication. In *2007 IEEE 65th Vehicular Technology Conference-VTC2007-Spring*, pages 21–25. IEEE, 2007.

[20] A. Jain, J. Mao, and K. Mohiuddin. Artificial neural networks: A tutorial. *Computer IEEE*, (3):31–44, 1996.

[21] N. Kohl. *Learning in fractured problems with constructive neural network algorithms*. PhD thesis, The University of Texas at Austin, Austin, TX 78712, 2009.

[22] N. Kohl and R. Miikkulainen. Evolving neural networks for fractured domains. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1405–1412. ACM, 2008.

[23] N. Kohl and R. Miikkulainen. Evolving neural networks for strategic decision-making problems. *Neural Networks*, 22(3):326–337, 2009.

[24] K. Korosec. Tesla: This is our most significant step towards safe self-driving cars, 2016.

[25] R. Miikkulainen. Neuroevolution. In *Encyclopedia of Machine Learning*. Springer, New York, 2010.

[26] M. A. Potter and K. A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation*, 8(1):1–29, 2000.

[27] J. Schrum and R. Miikkulainen. Constructing complex npc behavior via multi-objective neuroevolution. *AIIDE*, 8:108–113, 2008.

[28] J. Semarak. Intelligent traffic lights control by fuzzy logic. *Malaysian Journal of Computer Science*, 9(2):29–35, 1996.

[29] K. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

[30] K. Stanley, D. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[31] K. O. Stanley. *Efficient evolution of neural networks through complexification*. PhD thesis, The University of Texas at Austin, Austin, TX 78712, 8 2004.

[32] T. van den Berg and S. Whiteson. Critical factors in the performance of hyperneat. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 759–766. ACM, 2013.

[33] C. Yang and T. Morton. Trends of transportation simulation and modeling based on a selection of exploratory advanced research projects: Workshop summary report. Technical report, 2012.

[34] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

APPENDIX

A full appendix is available online[4].

[4]https://github.com/Amposter/Unity-AIM/tree/master/Appendices