UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

DEPARTMENT OF COMPUTER SCIENCE

COMPUTER SCIENCE HONOURS FINAL PAPER 2016

Title:	Intersection Management of Autonomous Vehicles Using NEAT and other Techniques
Author:	Jonathan Schoeman
Project abbreviation:	DSDC
Supervisor:	Geoff Nitschke

Category	**Min**	**Max**	**Chosen**
Requirement Analysis and Design	0	20	-
Theoretical Analysis	0	25	25
Experiment Design and Execution	0	20	-
System Development and Implementation	0	15	-
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

Neural Evolution versus Heuristic Control for Autonomous Vehicle Intersection Management

Jonathan Schoeman
Department of Computer Science
University of Cape Town
Cape Town, South Africa
Email: jonathan.schoeman.za@gmail.com

Abstract— ***In this paper we look specifically at using Neural Evolution to develop decentralised control for potential future traffic intersection management scenarios. We then compare this approach with an established centralised control method. [Complete abstract!]***

1. INTRODUCTION

Over recent decades huge advances in the field of machine learning have been made that have paved the way for new and exciting applications that were not possible in the past. One such application is that of intelligent and robust self-driving vehicles [17, 6, 5]. Creating such a controller for an autonomous vehicle would require a sophisticated "Artificial Intelligence" that is able to react dynamically to new situations in real time. Using modern evolutionary computing techniques such as Neural Evolution for Augmenting Topologies (NEAT) [18] and Hypercube-based Neural Evolution for Augmenting Topologies (HyperNEAT) [19] or variations thereof to evolve vehicle controllers that are capable of autonomously navigating roadways has already proven to be largely successful [5, 6, 17, 1, 9, 7]. In this paper we focus on comparing different techniques for creating simulated vehicles (agents) that are effective at navigating populated intersections of varying complexity. Specifically we look at autonomous sensory based agents evolved through NEAT and HyperNEAT and how they compare to another popular heuristic approach known as Autonomous Intersection Management (AIM). AIM has proven to be effective at allowing vehicles to navigate efficiently through simple densely populated intersections while avoiding collisions [8]. AIM represents a centralised solution which inherently has some advantages over decentralised solutions such as sensory based agents evolved through neural evolution. At any given time the AIM system is aware of all incoming vehicles and their respective positions, speeds and trajectories and can relay the required information to each vehicle in turn allowing them to navigate the intersection without any collisions with 100% certainty. However this approach requires each intersection to have a pre-installed intersection manager server as well as all vehicles navigating the intersection to contain an AIM transceiver. There are many reasons why this pre-installed infrastructure may be infeasible or undesirable for large scale implementation. To that end we aim to see how a self contained decentralised solution, namely a sensory based autonomous vehicle controllers evolved with an implementation of NEAT and HyperNEAT, compares with regard to performance and collision avoidance when navigating a variety of predetermined intersection types.

1.1 Research Objectives

The main objective of this experiment is to determine which of the techniques mentioned above can be utilized to achieve the highest performing agent for navigating populated intersections for each of the ten intersection types we are testing. We hope to find empirical evidence that a decentralized sensory based agent evolved with NEAT or HyperNEAT may achieve similar or better performance than an agent making use of a centralised heuristic approach such as AIM when navigating certain intersection scenarios. [Note that the metrics for measuring agent performance are discussed in section 3 below.] Based on previous research it is expected that HyperNEAT which utilizes an indirect encoding will converge faster to a more optimal solution than NEAT which uses a direct encoding [18, 19]. Our research objectives are stated plainly below:

Objective 1: *To determine if agents evolved with neural evolution can outperform agents making use of AIM for one or more of the ten predetermined intersection types of varying complexity.*

Objective 2: *To determine if agents evolved with neural evolution using an indirect encoding (HyperNEAT) outperform agents evolved with neural evolution using a direct encoding (NEAT) for one or more of the ten predetermined intersection types.*

2. BACKGROUND

This section introduces some of the concepts and algorithms used when conducting our investigation.

2.1 Neural Evolution & Controller Design

Neural Evolution (NE) is a machine learning technique that combines two prominent biologically inspired methods, that is, artificial neural networks (ANNs) and evolutionary computing (EC).

ANNs are computational systems of interconnected information-processing nodes (neurons) that are inspired by biological nervous systems [16]. As can be seen in Figure 1 a neuron can have up to any number of numerical inputs each with its own associated weighting. The weighted inputs are summed and applied to the neurons selected activation function which produces a single output value.

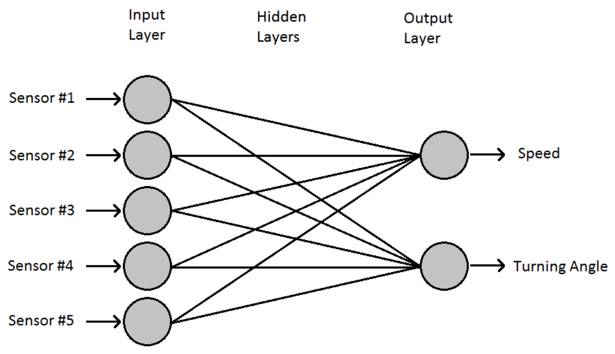

Figure 1: *The above figure shows an example of what a minimal fully connected ANN Topology used to initialize NEAT may look like for a basic vehicle controller that uses 5 sensors.*

These neurons can be connected together to form an ANN that depending on its topology can map up to any number of inputs onto any number of target outputs. Theoretically an ANN can be used to approximate any multi-variable function if the right topology and connection weights are used in combination [3]. An example of an ANN topology can be seen in Figure 2. ANNs are a promising candidate to use as a basis for our agent controllers as they have the potential to map numerical values from sensors into the required values that can be used to adjust an agents actions to avoid collisions in potentially any situation.

Evolutionary computing is a sub area of machine learning that bases its foundation on the the Darwinian principles of natural selection and adaptation. A population of ANNs, encoded as chromosomes, are evolved for a specific task using a type of evolutionary algorithm called a genetic algorithm [10]. Each ANN receives input to its network from information observed from the environment and computes some output. The ANN is evaluated and assigned a fitness

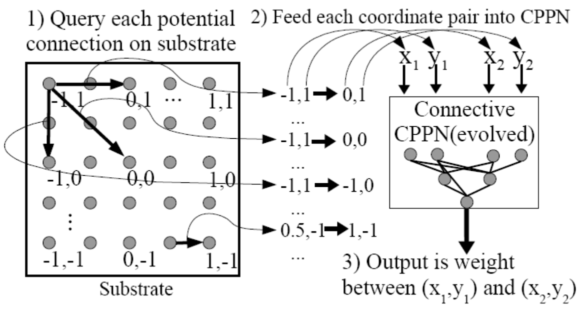
1) Query each potential connection on substrate
2) Feed each coordinate pair into CPPN
x_1 y_1 x_2 y_2
Connective CPPN(evolved)
-1,1 0,1 1,1
-1,0 0,0 1,0
-1,-1 0,-1 1,-1
Substrate
-1,1 → 0,1
-1,1 → 0,0
-1,1 → -1,0
0.5,-1 → 1,-1
3) Output is weight between (x_1,y_1) and (x_2,y_2)

Figure 2: *Hypercube-based Geometric Connectivity Pattern Interpretation* [19]: *The layout of nodes in space, called the substrate, is positioned and assigned co-ordinates 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 connection weight between the queried nodes.*

value using a fitness function that measures how well it performs on the specified task. ANNs that perform well are allowed to mate and propagate their genetic material into future generations.

2.2 NEAT

NEAT is a NE method that makes searching for solutions in high-dimensional space feasible by evolving ANNs. NEAT uses three fundamental principles that ensures the efficient evolution of the topology of complex networks and weights, 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 solves many of the problems present in general NE [4, 18] allowing it to evolve ANNs that perform well in control and decision-learning problems [18, 4, 21].

NEAT initializes individuals in a population with a minimal configuration (see Figure 1) 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 mate, NEAT uses a global counter to historically mark genes as they appear. This way, compatible parts of ANNs with differing topologies can be determined during recombination to result in meaningful offspring.

It should also be noted that NEAT employs direct encoding meaning there is a direct mapping between genes in the genomes and the nodes and connections they represent in their corresponding ANN phenotype.

2.3 HyperNEAT

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 to a second space that contains many. HyperNEAT is an extension of NEAT that instead employs an indirect encoding approach. It does this by evolving a population of connective compositional pattern-producing networks (connective CPPNs). These CPPNs are in turn responsible for creating ANNs that are evaluated on the specific task [7, 19].

CPPNs are abstractions of natural development. They are capable of producing complex patterns and regularities, much like those prevalent in nature, by composing simple functions. Using HyperNEAT, we first have to specify an initial *substrate configuration.* 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. Figure 2 shows an example of a substrate and connective CPPN. 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. 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.

2.4 AIM

The described goal of AIM is to create a scalable, safe, and efficient multi-agent framework for managing autonomous vehicles at intersections. AIM is a means for autonomous vehicles to communicate and to efficiently schedule and organise themselves when travelling through intersections so as to minimise waiting times [8]. AIM typically works by having autonomous vehicles wirelessly communicate with an intersection management server installed at each intersection. The intersection manager processes all intersection requests and tries to allocate the passing vehicles an optimal time and speed to cross the intersection. Heuristic control methods are the most commonly used approach for the scheduling process. This heuristic approach to autonomous vehicle controller design has proven to be effective [8] and will act as the standard to which we will compare the controllers produced using neural evolution.

3. METHODS

This section serves as an outline of the experiments we conducted. It also contains the implementation details of the simulation environment and each of the algorithms being compared.

3.1 Simulation Environment

The simulation environment for the experiment was built using the Unity3D[1] game engine. The simulation contains a total of 10 tracks and each track contains only one of the ten predefined intersection types. All the 3D meshes used for visually representing each track were built using Blender[2] a free and open source 3D modeling application. Each track was designed to represent a different scenario of varying complexity to comprehensively test the capabilities of each of the controller techniques being compared.

Track	**Intersection Type**
1	Simple T-Junction
2	Four Way Intersection
3	Four Way Traffic Circle
4	Double Lane Merge
5	Double Lane Four Way Intersection
6	Triple Lane Four Way Intersection
7	Offset Four Way Intersection
8	Custom Intersection (Princess Ann)
9	One Way Merge With Connecting Road
10	Custom Intersection (Newlands Avenue)

Figure 3: *Table showing intersection types for each track. For a full list of detailed images of track layout please refer to Appendix A*[4].

With track 8 and 10 being modelled off real intersections of interest surrounding the University of Cape Town. For a detailed list of all track layouts please see Figure 3.

A track is defined by a collection of track waypoints that are connected by mathematically defined bezier curves [11]. The bezier curves represent the path that an agent can follow to navigate between two specific waypoints. There are four types of track waypoints that can be defined. The *start_node* (green), the *end_node* (red), the *normal_node* (blue) and the *intersection_border_node* (cyan).

The *start_node* represents an entry point into a track and is used as a position to spawn in new agents into the simulation. Similarly an *end_node* represents an exit point of a track and is used to remove agents from the simulation. A *normal_node* is used to bridge two nodes that are difficult to connect with a single bezier curve. Finally the *intersection_border_node* is used to represent the edge of an intersection which in this case is defined by having

[1]https://unity3d.com/ - Date Accessed: 2016-11-01
[2]https://www.blender.org/ - Date Accessed: 2016-11-02

[4]Appendix A: https://github.com/Amposter/Unity-AIM/blob/master/Appendices/Tracks/Tracks.md

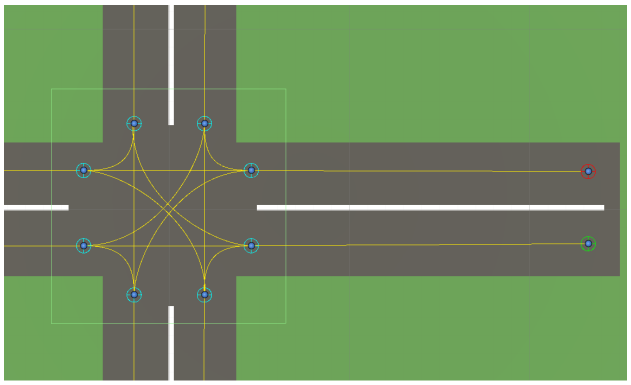

Figure 4: *The above image is a subsection of track 2 and serves as an example of how a simple four way intersection may be defined using the various track waypoints. The green square surrounding the intersection represents the trigger volume for agents using the AIM protocol to query the intersection management server of the intersection.*

two or more paths connected to it or two or more paths leading from it. When an agent is spawned at a specific *start_node* a random *end_node* is selected from a list of all *end_nodes* associated with the current track and an implementation of *Dijkstra's algorithm* [13] is used to find the shortest path connecting the two nodes. This path is then passed to the agent as a list of bezier curves and represents its desired route through the track. Since the research focus is automated intersection management it is assumed that the agent uses navigational data to navigate while managing its own speed in order to avoid obstacles similar to how AIM is implemented.

3.2 Evolutionary Controller Design

The NEAT and HyperNEAT evolutionary algorithms used for the experiment were implemented using the UnityNEAT[5] API, an extension of SharpNEAT[6], which has been modified to work with the Unity game engine. SharpNEAT is an open source framework for creating experiments involving neuro-evolution using the C# programming language.

In order to evaluate our ANN based controllers we created agents which serve as virtual vehicles within the simulation and act as an interface between the controller and the virtual environment. Each agent is equipped with sensors that detect the distance to the nearest obstacle along a straight path. Ten distinct sensors are used for each agent so that the ANNs have the potential to derive the distance to as well as the relative positions of obstacles based on the readings and orientation of individual sensors. The sensory configuration used for each agent can be seen in Figure 6.

While an agent is being evaluated readings from its ten sensors are normalized to values between 0 and 1. Where a value of 1 means there is an obstacle directly in front of the sensor and a value of 0 means there is no obstacle within sensor range. These normalised sensor readings are then used as inputs to the agents ANN controller. The ANN then computes a single value between 0 and 1 to be used to adjust the agents speed along its set path where a value 0 is stopped and a value of 1 is maximum speed. Basic agent parameters used in the simulation can be seen in Figure 6. Parameters were chosen to balance accuracy and performance while running the simulations at increased timescales.

3.2.1 The Fitness Function

In order for an agent's performance to be maximised its performance needs to be defined as a quantifiable value. This performance metric is known as an agents fitness. The fitness function is a mathematical construct common to all neuro-evolution algorithms that is used to compute the fitness value of an individual within the population based on a number a parameters. The desired learnt behaviour of an agent is encoded within the fitness function used to evaluate it. With this in mind we created a function that tries to maximise average speed while at the same time avoiding collisions. The fitness function can be seen below:

[5]https://github.com/lordjesus/UnityNEAT - Date Accessed: 2016-10-02

[6]https://github.com/colgreen/sharpneat - Date Accessed: 2016-10-02

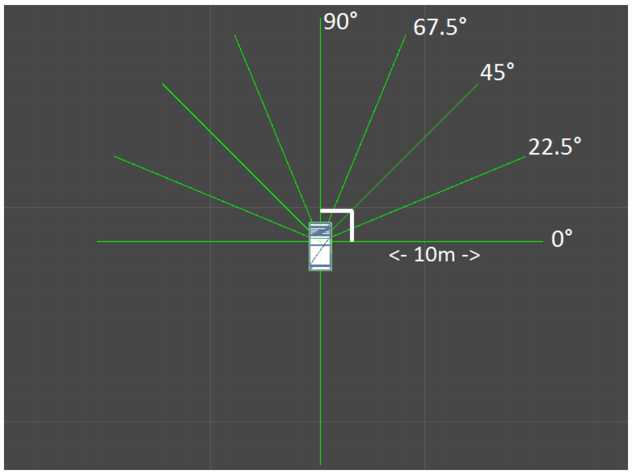

Figure 5: *The above image shows the sensory configuration used for each agent. A total of ten sensors are used each measuring distance to the closest obstacle with a maximum range of 10 metres.*

Agent Parameters	
Maximum Speed	8 m/s
Number of Sensors	10
Sensor Range	10m
Sensor/Controller Update Frequency	15 Hz

Figure 6: *Table showing basic agent parameters used during simulation.*

$$F = max(1, (S*2000)+(M*1000)-(c*250)-(b*100))$$

F = value between 1 and 3000 representing the final fitness of the controller.

S = average speed of all agents being evaluated over the duration of the trial normalised to a value between 0 and 1.

M = 1 minus the average distance to the nearest obstacle of all the agents being evaluated over the duration of the trial normalised to a value between 0 and 1.

c = number of agent collisions over the duration of the trial.

b = number of times a new agent was unable to spawn because there was another agent blocking the spawn region. (This penalty was introduced to stop agents from wasting time trying to increase fitness by blocking spawn points and maximising **M**)

Due to the way agents select random paths when they are spawned and the random time delay between each spawn it is possible for controllers to achieve an unusually high or low fitness that does not reflect the true potential of the controller itself. This may happen when the the agents converge at an intersection in such a way that collisions are more or less probable. For instance when the majority of agents select paths that results in sparsely populated intersections controllers may achieve an unusually high fitness due to the low probability of collision. To mitigate this phenomena every time a controller is evaluated multiple trials are run and the obtained fitness values averaged to obtain a final fitness value for the controller which more accurately reflects the controllers true potential.

3.2.2 Experimental Method

This section details the methodology used in conducting this investigation. For the purposes of this paper this section will focus primarily on the NEAT algorithm. The experiment was divided into three distinct parts and conducted separately. Those parts being AIM, NEAT and HyperNEAT.

For the AIM segment of the experiment a simplified version of the First Come First Serve AIM (FCFS-AIM) [8] protocol was implemented. As we had to re-implement AIM from scratch within the Unity framework certain parts of the protocol were generalised, the most important being that we did not discretise our tracks into individual cells that can be used to make reservations. Instead, we used an agents collision volume and trajectory to make reservations in three dimensional space. Furthermore, a few heuristics were added to prevent lane-overflow. Experimen-

High-level parameters	
Simulation Runs	20
Number of Trials	3
Trial Duration	40 sec
Population Size	150
Generations	150
Species	5
Total Agents per Trial	48
Agent Spawn Interval	1.1 - 3 sec
Low-level parameters	
Fixed Input Nodes	10
Fixed Output Nodes	1
Activation Function	Steepened Sigmoid
Add Node Probability	0.001
Add Weight Probability	0.01
Delete Weight Probability	0.001
Weight Mutation Probability	0.888
Disjoint Excess Genes Recombination Probability	0.1
Selection Operator	Roulette Wheel
Activation Scheme	Acyclic
Complexity Regulation Strategy	Absolute
Complexity Threshold	33

Figure 7: *The above table shows a list of NE and experimental parameters used when evolving agents for both NEAT and HyperNEAT for each experiment.*

tal tests were run on each of the ten tracks separately. Each test consists of twenty runs each with a duration of 90 seconds. During each run agents running the AIM protocol are spawned at *start_nodes* at random intervals of 1.1 to 3 seconds provided there are no other agents blocking the area. A maximum of plus minus 48 agents depending on the number of *start_nodes* are allowed to spawn per run, enough to populate each intersection while still allowing for the simulator to run with acceptable performance at higher timescales. When an agent is spawned it selects a path to a random *end_node* and attempts to navigate to it relying on AIM to prevent collisions with other agents while moving through inter-

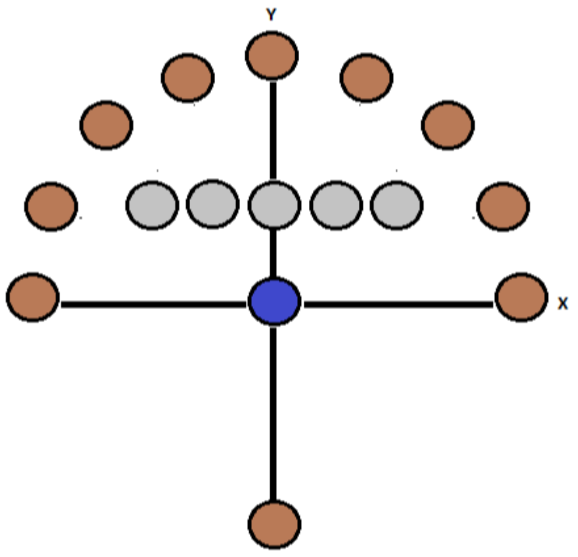

Figure 8: *The above image shows the substrate configuration used for the HyperNEAT algorithm. Where the orange circles represent the inputs, the grey circles represent the initial hidden nodes and the blue circle represents the output node.*

sections. If a collision is detected the colliding agents are immediately removed from the simulation and the collision is recorded. If an agent reaches its target *end_node* it is also removed from the simulation. At the end of the 90 second run all agents are removed from the simulation and the percentage of spawned agents that reached their destination, average agent speed, number of collisions and total agent idle time are recorded.

For the NEAT segment of the investigation we first evolved controllers individually for each of the ten tracks using the NEAT algorithm. For the evolution stage of the experiment a population of 150 individuals was evolved over 150 generations. These parameters where selected to allow for sufficient genetic diversity as well as sufficient time for useful behaviours to emerge while still ensuring the evolution process would fit within the time constraints of our experiment. Please refer to Figure 7 for a detailed list of the parameters used for evolution process. The same

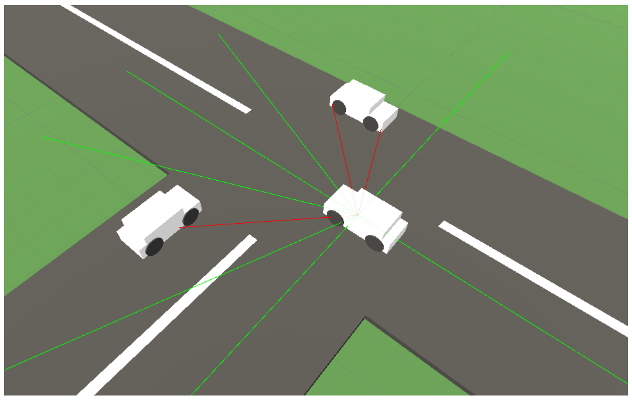

Figure 9: *The above image shows a preview of the simulation running as agents attempt to navigate a basic T-junction intersection. The right most agent's sensor readings are visually represented as red and green lines. Where red lines indicate an obstacle within sensor range.*

rules as the AIM agents with regard to agent spawning and collisions where applied to the NEAT agents during evolution. Each individual was evaluated using the fitness function discussed in section 3.2.1 over three trials with a duration of 40 seconds each. This process was repeated twenty times for each of the ten tracks. The highest performing controller out of the twenty runs of each track was selected to be tested under the same conditions used to test the AIM controller.

Since HyperNEAT is an extension of NEAT the same process and parameters where used when evolving and testing the HyperNEAT agents. The HyperNEAT substrate configuration used during evolution can be seen in Figure 8.

4. RESULTS

This section highlights a summary of the results obtained from the experiments outlined in the previous section. Figure 10 shows the mean throughput over the 20 test runs of evolved agents vs agents utilizing FCFS-AIM for each of the 10 tracks. Figures 12 and 13 show the mean agent speed and total agent idle time respectively over the 20 test runs. Figure 11 shows the mean collision count over the 20 test runs for NEAT and HyperNEAT agents, as expected there were zero FCFA-AIM agent collisions and so the AIM collision counts were omitted from the figure. For the full list of results obtained for AIM, NEAT and HyperNEAT please refer to Appendix C[7].

An example of a champion ANN topology that was evolved during the experiment using NEAT can be seen in Figure 14. A full list of ANN topology visualisations of all champion controllers used for testing for both NEAT and HyperNEAT can be found in Appendix B[8].

5. DISCUSSION

As expected the results show that the centralised heuristic agents (FCFS-AIM) perform well across all tracks with consistently high throughput and no agent collisions. FCFS-AIM also saw a very consistent mean agent speed for all intersection types with the exception of track 9 as can be seen in Figure 12. This dip in mean agent speed for track 9 is observed for both centralised (FCFS-AIM) and decentralised (NEAT, HyperNEAT) controllers and can be explained by this track having unusually high agent density at intersections. This can be attributed to the layout of track 9 having 7 start nodes and only 1 end node causing all traffic flows to converge at a single point (The layout of track 9 can be seen in Appendix A).

It is interesting to see that decentralised agents evolved through NEAT were able to achieve similar performance on some tracks when compared to the centralised heuristic agents even with their limited knowledge of the environment. Specifically NEAT performed well on tracks 3, 4, 5 and 9 with very high throughput and zero collisions. For track 3 NEAT was even able to outperform FCFS-AIM by maintaining 100% throughput and a significantly higher mean agent speed with significantly lower agent idle time.

[7]Appendix C: https://github.com/Amposter/Unity-AIM/blob/master/Appendices/Results/Results.md

[8]Appendix B: https://github.com/Amposter/Unity-AIM/blob/master/Appendices/ChampionTopologies/Toplogies.md

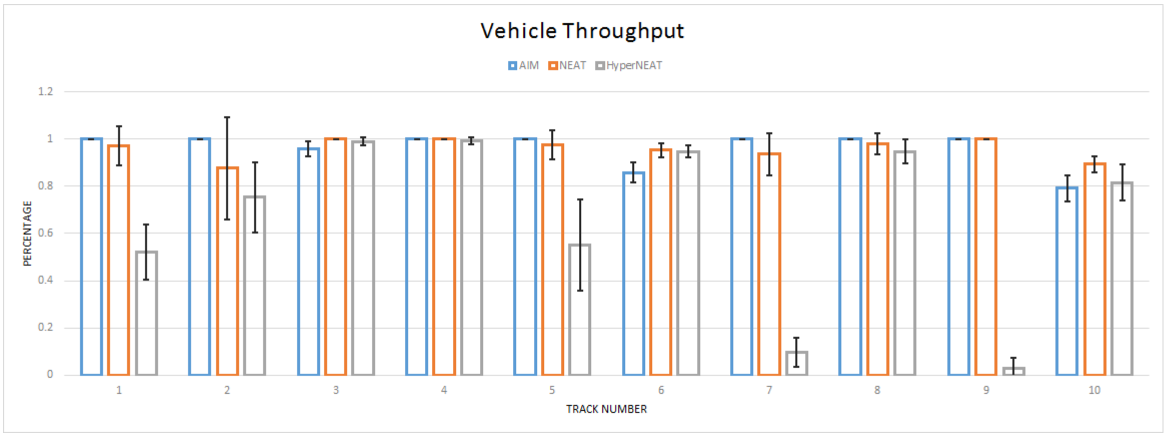

Figure 10: *The above figure displays the percentage of spawned agents that successfully arrived at their destinations within the trial duration, averaged over the 20 test trials. The black lines indicate one standard deviation above and below the mean.*

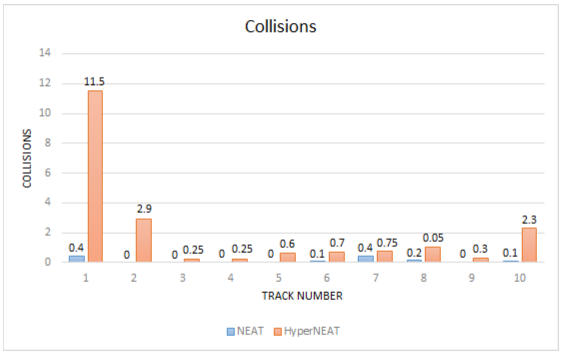

Figure 11: *The above figure displays the amount of agent collisions per track of NEAT and HyperNEAT agents, averaged over the 20 test trials.*

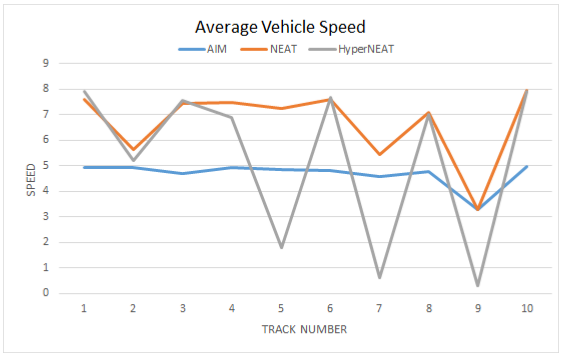

Figure 12: *The above figure displays the mean agent speed per track in meters per second, averaged over the 20 test trials.*

Another interesting observation is that although agents evolved through NEAT were able to perform well for some tracks agents evolved through HyperNEAT consistently performed poorly for every track. HyperNEAT agents experienced multiple collisions across all tracks and were prone to causing grid lock as can be deduced by the low throughput values and extremely high agent idle times seen in Figures 10 and 13. This suggests that NEAT was able to outperform our implementation of HyperNEAT within the first 150 generations of evolution for this predetermined task. This at first seems unusual based on previous research [15, 12]. However the poor performance of HyperNEAT can be explained by the experimental setup used and the nature of the learning task. HyperNEAT has shown to perform very poorly for tasks with irregular problem domains even when compared to NEAT [2, 20, 14]. The extreme case of an irregular problem is known as a fractured problem which can be defined, in this case, as a problem where

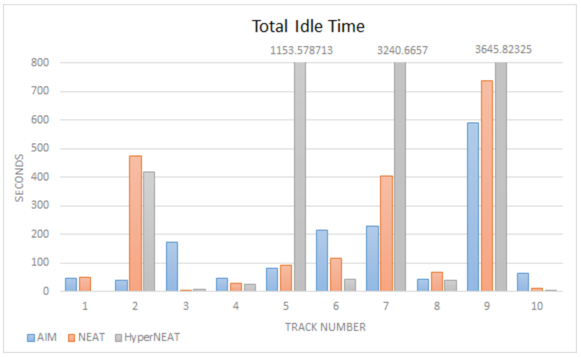

Figure 13: *The above figure displays the total idle time of all agents per track, averaged over the 20 test trials. (For ANN controllers agents with speeds of less than 0.1 m/s were considered idle.)*

the correct action varies discontinuously as the agent moves from state to state. Since the decentralised agents had no knowledge of their current trajectories the task posed in this experiment could be considered a fractured problem. For example if an agent detected an obstacle to its left and its current trajectory veered to the left the correct action would be to stop. However if an agent detected an object to its left and its current trajectory veered to the right the correct action may be to continue forwards. This results in a possible discontinuity in the correct action of an agent even if its sensor inputs remain the same.

This phenomena may also help to explain why NEAT and HyperNEAT performed poorly on some tracks and not on others. The results show that for track 3 NEAT was able to outperform FCFS-AIM in all fields however HyperNEAT also achieved good performance on this track save for a few collisions. Track 3 represents the single lane traffic circle intersection type. For this problem agents travel in a clockwise direction around the traffic circles and can only ever turn left while only ever having to give way to agents approaching from the right. This means that the task domain of track 3 is also the least irregular than that of other tracks perhaps suggesting the regularity of the task domain unique to each track effects the performance of the centralised agents.

Overall the results show that decentralised agents

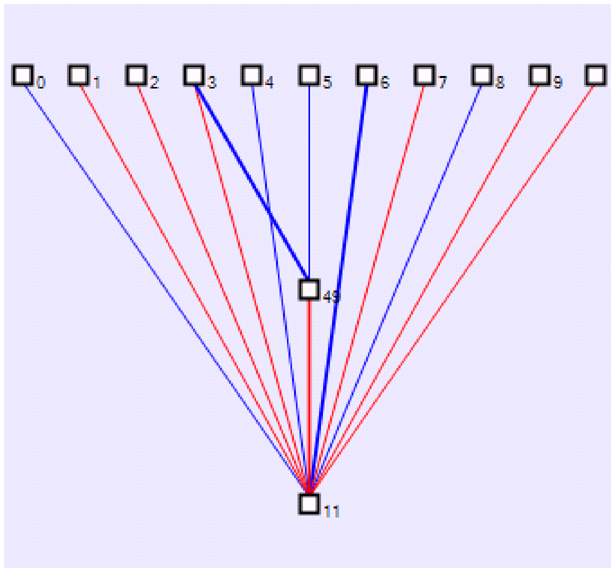

Figure 14: *The above figure displays the ANN topology that was evolved using NEAT for the highest performing controller of track 9. Nodes 0 to 8 are the front sensor inputs from left to right. Node 9 is the back sensor input and the right most node is the bias. Node 11 is the final speed output node for the controller.*

specifically those evolved using NEAT were able to achieve similar performance than that of centralised agents for 4 out of 10 of the predetermined intersection types even with the limited population size and number of generations used. This suggests that perhaps given more time to evolve more sophisticated behaviours, or given more information about the environment such as agent trajectory information, centralised agents may be able to achieve performance similar to that of the decentralised agents for all tracks.

6. CONCLUSION

-conclusions

References

[1] A. Agapitos, J. Togelius, and S. M. Lucas. Evolving controllers for simulated car racing using object oriented genetic programming. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1543–1550. ACM, 2007.

[2] 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.

[3] G. CYBENKO. Approximation by superpositions of a sigmoidal function. In *Mathematics of control, signals and systems 2*, pages 303–314, 1989.

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

[5] J. Drchal, O. Kapral, J. Koutník, and M. Šnorek. Combining multiple inputs in hyperneat mobile agent controller. In *Artificial Neural Networks–ICANN 2009*, pages 775–783. Springer, 2009.

[6] J. Drchal, J. Koutník, et al. Hyperneat controlled robots learn how to drive on roads in simulated environment. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1087–1092. IEEE, 2009.

[7] J. Drchal, J. Koutník, et al. Hyperneat controlled robots learn how to drive on roads in simulated environment. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1087–1092. IEEE, 2009.

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

[9] F. J. Gomez. *Robust non-linear control through neuroevolution.* PhD thesis, The University of Texas at Austin, Austin, TX 78712, 8 2003.

[10] F. J. Gomez and R. Miikkulainen. Active guidance for a finless rocket using neuroevolution. In *Genetic and Evolutionary Computation—GECCO 2003*, pages 2084–2095. Springer, 2003.

[11] W. J. Gordon and R. F. Riesenfeld. Bernstein-bézier methods for the computer-aided design of free-form curves and surfaces. In *Journal of the ACM (JACM) 21.2*, pages 293–310, 1974.

[12] R. T. P. Jeff Clune, Charles Ofria. The sensitivity of hyperneat to different geometric representations of a problem. In *GECCO '09 Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 675–682, 2009.

[13] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. In *Journal of the ACM (JACM) 24.1*, pages 1–13, 1977.

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

[15] S. G. Lowell, Jessica and K. Birger. Comparison of neat and hyperneat performance on a strategic decision-making problem. In *Genetic and Evolutionary Computing (ICGEC)*, 2011.

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

[17] D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. In *Neural Computation 3.1*, pages 88–97, 1991.

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

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

[20] 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.

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