# EVOLVING CONTROLLERS FOR SIMULATED CAR RACING USING NEUROEVOLUTION

by

**Aravind Gowrisankar, B.E.**

**THESIS**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**MASTER OF ARTS**

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2008

# EVOLVING CONTROLLERS FOR SIMULATED CAR RACING USING NEUROEVOLUTION

To Amma and Appa for putting my education before everything else

# Acknowledgments

I would like to thank Risto Miikkulainen for his patient support during all stages of this thesis. Risto's Neural Network class inspired me to start doing research in neuroevolution.

Thanks to Ugo Vieruchi for coding JNeat and Julian Togelius for creating the simplerace domain which I used as the base to write the simulations. I also want to thank the Neural Networks group for their valuable suggestions towards this project.

The masters program at UT has exposed me to education and research of the highest quality. I am grateful to the CS instructors who led me through the masters program. I found Glenn Downing, Greg Plaxton, Peter Stone, Ray Mooney and Risto Miikkulainen to be inspirational instructors and I cherish my experiences from their classes and lectures.

Being a graduate student in CS has slso given me a chance to work on class projects in inter-disciplinary fields like Bioinformatics. I consider myself lucky to have worked with Andrew Ellington and Edward Marcotte at the Institute of Cellular and Molecular Biology. I thank them for giving me an opportunity to work on exciting projects.

I am grateful to John McDevitt and Pierre Floriano for providing me with an opportunity to be a graduate research assistant during my masters program. The McDevitt group helped me acclimatize to the new environment I faced when I came to the US. They also gave me a chance to apply my CS skills to projects with a positive social impact. I must also thank Margaret Myers and Robert Van de Geijn for their kindness and hospitality.

I am also thankful to Maytal Saar Tsechansky for providing me with an opportunity to be a Teaching Assistant for the Data Mining course. I am extremely happy to have taken a class with Jeffrey Martin which opened my eyes to the world of entrepreneurship.

The Technology Entrepreneurship Society has proved to be a wonderful avenue for interacting with peers from other fields and programs, notably engineering and business students. My experiences working with fellow TES officers and organizing various events have been fun.

I consider myself lucky to have worked with graduate students like Sindhu Vijayaraghavan, Sudheendra Vijayanarasimhan and Venkat Balachandran on interesting projects. I also thank my friends for their encouragement and the help that they gave me at different moments during these two years: Ashwin Parthasarathy, Ashwin Radhakrishnan, Bakhtiyar Uddin, Easwar Swaminathan, Mario Guajardo, and Sudheendra Vijayanarasimhan. I treasure the memories from the wonderful experiences I have had with them.

I am forever grateful to my parents for their love, support, and the sacrifices they have made over the years. I also owe special thanks to my family(aunts and uncles) for their love and support. I thank my cousins in the US for their kindness and hospitality; I have enjoyed the time spent with their families. Last, but definitely not the least, I am thankful to the love and encouragement from my wife Dhivya.

**EVOLVING CONTROLLERS FOR SIMULATED CAR RACING USING NEUROEVOLUTION**

Aravind Gowrisankar, M.A.
The University of Texas at Austin, 2008

Supervisor: Risto Miikkulainen

Neuroevolution has been successfully used in developing controllers for physical simulation domains. However, the ability to strategize in such domains has not been studied from an evolutionary perspective.

This thesis makes the following three contributions. First, it implements Neuroevolution using NEAT with a goal of evolving strategic controllers for the challenging physical simulation domain of car-racing. Second, three different evolutionary approaches are studied and analyzed on their ability to evolve advanced skills and strategy. Though these approaches are found to be good at evolving controllers with advanced skills, discovering high-level strategy proves to be hard. Third, a modular approach is proposed to evolve high-level strategy using Neuroevolution. Given such a suitable task decomposition, Neuroevolution succeeds in evolving controllers capable of strategy by using a modular approach. The simplerace car-racing simulation[29] is used as a testbed for this study. The results obtained in the car-racing domain suggest that the modular approach can be applied to evolve strategic behavior in other physical simulation domains and tasks.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Physical simulation domains serve as challenging testbeds for modern AI methods. Creating intelligent controllers for that are capable of strategy in these domains is hard. The goal of this thesis is to present and analyze ways to evolve controllers that possess advanced skill and strategy in physical simulation domains using Neuroevolution.

## 1.1 Physical Simulations and Computer Games

Physical simulations are very important for studying complex real world problems. They help researchers focus on their ideas and provide an accessible platform for conducting experiments. Simulations are useful because they provide a simple and tractable domain. On the other hand, real world application domains are often so complex that one can spend huge amounts of time on details that have little to do with research. Simulations make it easier for researchers to focus on the ideas rather than the intrinsic complexities and implementation issues present in the real world. Once the simulation is successful, a prototype can be built and tested in the real world.

Today, Computer and Video games are easily the most widespread simulations available and are used widely for AI research. Games capture people's imagination and offer inspiration for research. IBM's Deep Blue, Blondie24 (checkers) and Blondie25 (chess) are examples which have caught the attention of the public. Computer games of today encompass a wide variety of games including board games, action games, strategy games, role playing games, vehicle simulation games, etc. Each type of game offers a different challenge from an AI perspective. For example, in single player games, the player strives to reach a selfish goal. On the other hand, players in multi-player games, may have to work against other players (opponents) and/or work in cohesion with some players (team members). Team games require coordination between the team members to ensure that all members work towards the common goal.

Traditional computer games like board games are different from physical simulation domains because of the nature of the game environment. In board games, the environment is discrete and the number of possible actions and percepts are finite. Board games like Checkers, Othello and Chess have been used for AI research and applications like these led to the rise of Good Old Fashioned Artificial Intelligence (GOFAI). On the other hand, physical simulations model the real world and are continuous, i.e. the number of possible actions and percepts are infinite. Modern computer games and video games have continuous environments and hence can be used for physical simulations. Video games present an opportunity and challenge for computational intelligence methods just like symbolic board games did for GOFAI[13]. The eventual goal of such research is to transfer the knowledge learned from the physical simulation domain to the real-world e.g. using robots. Robocup soccer[19] is an example of a physical simulation domain where lessons learned in simulation have been transferred to real robot soccer competitions. Often, ideas based on research in a particular game have inspired work in completely different domains[18]. Previous research in physical simulation domains is discussed in Section 2.2.

## 1.2 The Car Racing Domain

Car racing is a good example of a physical simulation domain. It presents a lot of challenges for controller development. Some of them include:

1. Learning the Skills : The skills needed for car racing can be split into two categories : basic and advanced.

    - Basic : The controller needs to know how to accelerate/brake, steer and change gear. Without these skills it is not possible to be competent. These skills can either be programmed by hand or learned. Sometimes learning basic skills through AI methods can give rise to fine tuned skills.
    - Advanced : Advanced skills leverage the basic skills. They can be used to gain an edge over the opponents. Examples include overtaking, late-braking, learning to use the traction of the track, etc. It may take a significant effort to program advanced skills by hand.

2. Opponents : Adapting to an opponent is key for success in this domain. Some opponents may be conservative, some may be aggressive. Knowing such information (beforehand or recognizing it during the game) can help make better decisions, for example, while overtaking. Also, the controller should know to to defend its position if its under threat from an opponent.

3. Recovery : The controller should have a recovery mechanism. If the car goes off track or if there is a collision, the controller needs the ability to get back on track. In case of collisions, the car may be knocked out of the race or may need some repair (pit stop).

4. Strategy : Strategy is different from skill. A skill is an ability to do something competently by executing a sequence of actions. Strategy refers to a higher

level behavior like making a plan towards a goal. To implement a particular strategy, one or more skills may be needed. Car-Racing provides plenty of opportunities for strategy. One of the essential decisions to make is to estimating the chance of an overtake maneuver. Turns present good opportunities for overtaking. But the feasibility of such maneuvers should be decided on how the opponent is driving. Sometimes decisions need to be taken depending on the context. If the controller is at the last position it can afford to be aggressive; but if its in second position, it cannot afford to take unnecessary risks and give away its advantage. Real world racing (and even computer games) features pit stops, multiple laps and even multiple races (championship) all of which add to the strategy element. Timing the pit stops is often crucial in races.

5. Real Time Issues : Controllers have to make quick decisions in the car racing domain. It is not enough if the controller makes the right decision, the decision must be timely. Otherwise, an advantageous position maybe lost to an opponent. In real world car-racing, drivers must also be able to adapt to changing environment like rain.

Many of these issues arise in other physical simulations domains as well, but the impact of these issues are easy to observe and study in car-racing. This makes car-racing an ideal AI platform to studying development of skill and strategy. This thesis uses car-racing as a test-bed for developing controllers capable of intelligent behavior.

## 1.3 AI Methods for Games

For game playing domains, the Reinforcement Learning (RL) paradigm proves to be a good fit. They do not require training examples like traditional learning methods. This is important in game playing domains, as it is impossible for a human to provide accurate and consistent evaluations of a large numbers of positions, which would be needed to train an evaluation function from examples[24]. The main feature of RL algorithms is that they provide a mechanism to develop game controllers by experimentation. Successful moves, the corresponding skills and strategies are stored and continually refined by playing games repeatedly. These methods require some sort of feedback for their actions. Games typically come with a numeric score and a win/loss/draw result which can be used as the feedback. RL algorithms typically learn a value function that represents the intrinsic value of being in a particular state. Temporal Difference Learning(TD) is an example of a reinforcement algorithm that attempts to learn a value function by experimenting with different actions. Value function reinforcement learning algorithms like TD can solve problems without requiring examples of correct behavior. However, value function reinforcement learning methods have problems dealing with large state spaces[7] and hidden states[15] which characterize physical simulation domains. Further, for playing games with opponents, algorithms like TD need opponents to be defined beforehand (hand-coded or using other approaches). It is hard to create opponents that are conducive to learning.

Evolutionary Algorithms (EA) present an alternative approach to game playing. They are based on the principles of natural evolution. EA uses operators inspired by biological evolution: reproduction, mutation, recombination and selection. EA evolves a population of individuals. Each individual in the population represents a candidate solution. Like Reinforcement Learning, EA does not require examples of game situations; It needs a fitness function for evaluating the candidate solution in the environment. A fitness function is a numerical reward given to an individual based on its performance in the environment. After evaluating every individual from the population in the environment, the genetic operators are applied and the next population is created. The fittest individuals survive and reproduce. The process is repeated until a solution is obtained.

Evolutionary Algorithms have been used in a number of domains including game playing. Games are suitable testbeds for EA because every game results a numerical score (soccer) or a win/loss/draw (tic-tac-toe) result which can be used as a fitness function. Evolutionary Algorithms have also become popular for their ability to come up with novel solutions to complex real world problems like antenna design[8]. Such abilities can be very handy in the game playing domain, to evolve effective game playing strategies. Evolutionary Algorithms can also be used to evolve the opponents along with the game playing controllers. Such approaches, called Coevolutionary approaches, have been applied successfully to a wide variety of game playing domains [11], [14], [22].

Neuroevolution is a class of Evolutionary Algorithms that combine the power of Evolutionary Computation with Neural Networks. Neural Networks have been successfully used in a wide variety of problems ranging from classification to control tasks and regression. Their benefits including non-linearity, adaptivity, generalization and fault tolerance have been well documented. Despite being a popular and powerful learning method, the design of Neural Networks is considered difficult. The number of parameters that need to be configured including the inputs, the connections, the number of hidden layers, etc. makes the job of a neural network designer a difficult one. To complicate matters, a neural network that works perfectly in one domain, may not work in another domain. Hence neural network design is done using a combination of previous experience and trial and error.

Neuroevolution uses evolutionary principles to evolve the neural network instead of designing it by hand. Evolution starts out with a population of random networks. It uses a fitness function to evaluate the networks and applies the genetic operators to create the next population of networks. The Neuro Evolving Augmenting Topologies (NEAT) algorithm proposed by Stanley and Miikkulainen[28] is an example of Neuroevolution. It provides a mechanism to efficiently evolve Neural Networks through complexification. Using NEAT, the Neural Networks start minimally and grow in complexity (nodes,links) incrementally. Hence NEAT can be used to design the Neural Network instead of manually designing it. NEAT is described in Chapter 2.

Neuroevolution has been successfully used in developing controllers for a variety of tasks. Gomez and Miikkulainen[6] used Enforced Sub Populations for active finless rocket guidance, a physical simulation domain. In the gaming domain, the SANE Neuroevolution method has been used in evolving game players for board games like GO[11] and Othello. Coevolution using NEAT has been shown to be successful in General Game Playing[20]. Real Time Neuroevolution has been successfully used in evolving Non-player characters in the NERO video game[25].

As mentioned above, Neuroevolution methods have been studied and used to develop controllers for gaming domains and physical simulations, but these studies focussed only on the control aspect. Developing controllers capable of strategy is a harder problem and has not received the same amount of attention from the AI community. Games like Poker and Prisoner's dilemma[references] have been studied extensively from a strategy perspective. However physical simulation domains have not been a part of such studies. Is Neuroevolution capable of discovering novel strategies in such domains? In particular, can NEAT be used to evolve high-level strategies in physical simulation domains?

Of late, there has been a significant interest in game playing and numerous competitions have been taking place to encourage research in computational intelligence. Some of the competitions conducted include Ms. Pac-Man, Othello, X-Pilot AI and Simulated Car Racing. Simulated Car Racing is a two player car-racing game developed by Lucas and Togelius[29]. The domain used (aptly called 'simplerace') is a slightly simplified version of the car racing problem discussed above. However, it presents plenty of opportunity for evolving skill and strategy. In this thesis, the simplerace domain is used as the testbed for evolving skill and strategy using Neuroevolution.

This thesis makes three contributions. First, it implements Neuroevolution using NEAT on a physical simulation domain i.e. car racing. Second, three different evolutionary approaches are systematically studied and analyzed on their ability to evolve advanced skills and strategy. Third, a modular approach is proposed, evaluated and implemented to evolve high level strategy using Neuroevolution.

The conclusion of this thesis is that NEAT can be used to evolve controllers for challenging domains like car-racing. NEAT is shown to discover advanced driving skills without the aid of any domain knowledge. Discovering strategy and high level behavior is found to be much harder. To overcome this, some domain knowledge is used in decomposing the problem to relatively independent tasks. Once such a problem decomposition is setup, NEAT is able to evolve high level strategy. This modular approach can be applied to not only car-racing but physical-simulation domains in general. Eventually, the knowledge and strategies discovered by Neuroevolution from the simulations can be transferred to real-world domains.

# Chapter 2

# Background

The car-racing domain is an instance of the larger problem of developing controllers for physical simulation domains. The first section motivates neuroevolution and the suitability of NEAT for such domains. The second section describes the challenges associated with developing controllers for physical simulation domains and previous work in these domains. Finally the simplerace car-racing domain that is used as the testbed for this thesis, is introduced.

## 2.1 Neuroevolution

Traditional learning methods are supervised i.e. they require examples of situations that arise in the problem domain. The number of possible situations that arise in any computer game is extremely large even for board games; for continuous environment games, it is infinite. Evolutionary Algorithms (EA) present an attractive approach to game playing. Rather than learning from a set of examples, these biologically inspired methods learn by experimenting with different possible actions. Neuroevolution (NE) is a powerful evolutionary algorithm that has shown to be successful in a wide variety of domains including game playing. NE is a mechanism for constructing neural networks using evolutionary algorithms. Neural Networks have been used in a wide variety of control tasks and are a powerful method for capturing non-linearity in a domain. They can handle continuous states and inputs effectively. Further, the hidden neurons of a neural network together with the recurrent connections of the network can capture the hidden states that arise in a problem. Such recurrent neural networks can provide a non-linear map sensor inputs into an effective action. The evolutionary algorithm is used for evolving the structure and weights of the neural network.

Traditional Neuroevolution methods allow only the evolution of the connection weights of the neural network. The topology has to be designed in advance. A topology which works for one domain need not work for all domains. If the chosen topology does not match the problem at hand, evolution searches the wrong solution space and consequently cannot find a good solution. Hence a designer has to experimentally try out different configurations before selecting one.

NEAT[28] provides an elegant solution to this problem by evolving the topology of the network in addition to the connection weights. Two special operators are introduced in NEAT to i)add link between existing nodes and ii)create new nodes. The genetic operators to add nodes and links represent structural mutations. NEAT networks start minimally and expand during the course of evolution by using the various genetic operators. The expansion of the topology by adding new links and nodes allows NEAT to search higher dimensional search spaces. This ability to expand the dimensionality of the search space while preserving the values of the majority of dimensions is called complexification.

NEAT has three key features which make complexification possible.

1. Genetic Encoding and operations: The genetic encoding in NEAT is flexible and allows expansion. Each genome in NEAT has two sets of genes - node genes and connection genes. The node genes maintain information about the type of the node (hidden, input or output). The connection genes represent a link between two nodes. Each connection gene specifies the in-node, the

out-node and the weight of the connection. It also has an innovation number and an enable bit. The innovation number allows finding corresponding genes during crossover and the enable bit represents whether the connection gene is expressed or suppressed. Structural mutations are implemented using the connection genes and node genes. For an add-node mutation, an existing connection is split and a new node is placed where the old connection used to be. The old connection gene is disabled and two new connection genes are added. For an add-connection mutation, a new connection gene is added to connect two previously unconnected nodes. This flexible definition of genes is powerful and enables complexification of the networks during the course of evolution.

2. Tracking genes through Historical Markings: Genomes grow large during evolution because of add-node and add-connection mutations. Implementing crossover between two genomes of different lengths can be tricky. The innovation number stored in the gene acts as a historical marking and can be used to find matching genes. During crossover, genes with same innovation number are lined up and one of them is randomly selected for the offspring. Genes that are not matched are inherited from the more fit parent. These innovation numbers avoid the need for topological analysis and allow crossover to be performed efficiently.

3. Protecting Innovation through Speciation: New individuals with structural innovations cannot compete with the best of the population. They need at least a few generations to optimize their structure. To protect novel innovation, NEAT implements speciation. Individuals are grouped into species based on the similarity of their topologies. Again historical markings are used to find the similarity between two genomes. During reproduction, individuals com-

pete with other individuals within the same species and not with the entire population. As a reproduction mechanism, NEAT uses explicit fitness sharing. Organisms within the same species must share the fitness. This has a dual effect of ensuring that species do not become too big and structural innovations are protected.

NEAT networks start minimally with no hidden nodes. The three above principles ensure complexification and hence evolution can search a wide range of increasingly complex topologies simultaneously.

NEAT has been applied to a variety of hard reinforcement learning problems including pole-balancing and double-pole balancing. It has been used in the robot duel domain[28] and for playing pong[14]. NEAT has also been used for collision avoidance of vehicles[10]. The success of NEAT in such domains makes it an ideal choice for evolving game playing controllers for the task of car-racing.

## 2.2 Controllers for Physical Simulations and Car-Racing

Physical simulations of real world problems abound in the form of Computer and Video games. Physical simulations present a lot of challenges for developing controllers. The first challenge comes from the very nature of such simulations. Physical simulations are dynamic. The number of situations and scenarios that can arise in a physical simulation is very large. It is important for the controller to be able to adapt to various situations. Further, the input space and output space of a controller in such domains are continuous in nature and this makes the state space infinite. Effects of large state spaces can be alleviated to an extent by approximation. The harder problem is that arbitrarily small changes in the environment can make a huge difference in these domains.

The second big challenge is to develop controllers that can not only perform the task, but also possess advanced skills. Advanced skills are important, not only because they are interesting to watch (which is needed for gaming domains), but such skills are a sign of intelligent behavior. A harder problem is to play strategically. For example, bending it like Beckham (in soccer) is a skill. Wearing the opponent out in boxing like Muhammad Ali is strategy. Strategy can leverage any of the skills (including advanced skills), but clearly it is one step above skills in terms of intelligent behavior.

The third significant challenge is the need to adapt to opponents. Playing with an opponent opens up more possibilities for strategic play. Recognizing opponent's moves can help the controller gain an edge to counter them. Recognizing opponent's strategy can give the controller a stronger advantage to prepare a counter strategy. Opponent modeling is a challenging task and is a field in its own right[2].

Creating a controller that can do all the above can take significant programming work (if at all possible). Even before the implementation, just designing a controller which can do the above is a challenge. The way the inputs are presented to the controller (problem representation) can have a significant difference on the performance of the controller.

Physical simulations have been used by researchers to develop controllers with a goal of testing and applying various AI methods. The robocup soccer domain mentioned in Section 1.1 is one such domain. It has inspired research in multiple fields of AI, particularly multi-agent systems and reinforcement learning. In addition to capturing the challenges listed above, the soccer domain also requires communication between the various team members. Rocket navigation and real-world vehicle navigation are some other physical simulation domains that have been used for AI research. Gomez and Miikkulainen used Enforced Sub-Populations for finless rocket guidance in the rocket navigation domain[6]. Kohl and Miikkulainen used NEAT for developing real-world vehicle warning systems[10] to prevent collisions between vehicles.

In the studies mentioned above, the ‘control-aspect’ of physical simulation domains has been tackled successfully using NE methods. However, neither NE nor other methods have showed the ability to develop controllers capable of strategy automatically in any of these domains. In the past, some work has been done on developing strategic controllers for non-physical simulation domains. Bryant and Miikkulainen showed that NE can be used to develop visually intelligent behavior in the Legion II board game[1]. Evolutionary algorithms have been used for evolving game controllers for strategy games like Poker and Prisoner’s Dilemma. In [28], Stanley and Miikkulainen observed elaboration of behaviors when using NEAT in the robot duel domain. This elaboration was shown to be a benefit of complexification in NEAT. Though elaboration represents newly learned behavior, it does not imply strategic behavior. This is because strategy also involves selecting one of distinct multiple behaviors. So far, this has been hard to achieve using NEAT. Another physical simulation domain where high level decision making ability has been studied is keepaway. In [31], the authors developed a switch network to make high-level decisions for the keepaway soccer domain using three different methods: coevolution, layered learning and concurrent layered learning. Though the methods leveraged significant human expertise, they were found to perform worse than a hand-coded strategy in a hard-version of the keepaway task. In summary, previous research in physical simulation domains has been successful in dealing with the control aspect. However, developing strategy for physical simulation has been difficult for learning methods. This thesis uses Neuroevolution to study the strategy aspect in such domains.

Car-racing is a domain which not only captures the challenges listed above, but also permits easy observation of behaviors and strategies. Car-racing simulations are inspired from the real world counterparts i.e. human-driver car-racing. Human driven car-racing competitions like Formula-One have existed for over 50 years. Only recently have real-world races with driverless vehicles come into existence. The most popular and inspirational competitions are the DARPA Challenges which started in 2004. The DARPA Grand Challenge was a gruelling 150 miles race across the Mojave Desert and the main challenge was to adapt to different kinds of rough terrain. In the first instance of the challenge(2004), none of the cars finished the race and only five cars were able to complete the race in the second instance(2005). The third DARPA challenge was the Urban Challenge. Here the focus was to drive in an area that resembled a normal urban city. The challenge was to drive over 60 miles in the presence of other cars and follow the traffic lights, stop signs and negotiate obstacles. Six teams completed the entire course. Robotic Car Racing at the University of Essex is an ongoing autonomous car racing project. The cars are much smaller and comprise of a high end retail car, a laptop, a GPS receiver and camera. The goal here is to encourage teams to build autonomous racers using the same equipment. Though this competition is a scaled down version of the DARPA competitions, the challenges are almost the same. though at a much lower cost.

Many teams participate in driverless vehicle competitions. These competitions provide a learning platform that is accessible for researchers. Research in these platforms is important because driverless navigation is an important goal for the future. However for testing algorithms and comparing the strengths and weakness of different paradigms, simulation environments are preferable. Simulation environments like games, abstract some of the complexities that can arise in the real world and help focus on key research ideas. In this thesis, simulated car racing is used to study the ability of neuroevolution to develop controllers with skill and high-level strategy. Computational intelligence researchers stress on the fact that intelligence should be an emergent property[12]. This thesis works along a similar line of thought. The goal is to develop intelligent behavior in physical simulation domains without putting in much domain knowledge.

## 2.3 Simulated Car Racing in the Simplerace domain

The domain used as a testbed for this study is the simplerace package. Developed by Julian Togelius and Simon Lucas for the 2007 Car Racing Competition at the 2007 IEEE Symposium on Computational Intelligence and Games, the simplerace domain provides a platform for testing automatic controllers. In this domain, the quality of a controller is measured by the number of waypoints it can capture in a predefined time interval. The waypoints are randomly distributed around a square area and the controller knows the position of the current waypoint and the next. Waypoints can only be captured in the order of appearance, i.e. at any point of time, only the current waypoint can be captured. Though the next waypoint's position is known, it cannot be captured, but can be used to gain a strategic advantage over the opponent. A picture of the simplerace domain is shown in Figure 2.1.

In order to obtain a reliable estimate of a controllers performance from the simplerace domain, the average score obtained from five runs is used, where each run is a race with 1000 time steps.

### 2.3.1 Dynamics of the Simplerace domain

Though the simplerace domain is limited to a maximum of two players and does not consider some real-world issues associated with car-racing like wear and tear, the physics is fairly detailed. In the simplerace domain, the car is simulated

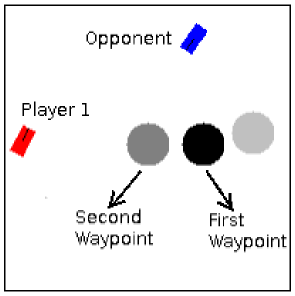

Figure 2.1: Simplerace domain. Player 1 and the opponent are marked in the figure. The dark black circle is the current waypoint; the gray circle is the next waypoint. The light gray circle is the third waypoint in sequence - it is not a part of the sensor model and is provided for visual cue only. The goal is to capture maximum number of waypoints(when driving solo) and defeat the opponent. The presence of an opponent and (randomly distributed) waypoints make simplerace a good testbed for studying evolution of strategy

as a $20 \times 10$ pixel rectangle, operating in a rectangular arena of size $400 \times 400$. The car's complete state is specified by its position, velocity, orientation and angular velocity. The state of the car and the simulation is updated 20 times per second. For more details including the dynamics of collisions, see [29]. Due to the dynamics of the simplerace domain(Appendix 1.2.2), the car accelerates faster and reaches higher top speeds when driving forwards rather than backwards. Also, the car has a smaller turning radius at low speeds and approximately twice as large turning radius at higher speeds due to skidding.

The races in the simplerace domain are essentially of two types. The first type is a single-car race. In this case the quality of the controller is indicated by the number of waypoints collected. The second is a two-car race - there are two cars on the track, meaning that a good controller will have to know how to get as quickly as possible to the current waypoint, and also defeat the other car on track by capturing more waypoints. In this case, the quality of the controller is indicated by the number of waypoints captured and the margin of victory over the opponent. Thus the domain serves as a convenient test-bed to test both skill (i.e. how fast the controller can travel, how sharp it can turn) and strategy (can I get there before the opponent?)

### 2.3.2 Features of the Simplerace domain

The representation of the environment and the input representation is an important step in solving the problem. The simplerace domain provides two kinds of sensors to get information about the current state of the car race. First-Person Sensors provide an egocentric representation of the world. The waypoints and the opponents are described by their distances and angles relative to the player. Third-Person Sensors on the other hand, provide absolute positions and velocities of the two players and the waypoints. A comprehensive listing of the sensors is provided in the Appendix 1.2.1.

A set of controllers are provided as a part of the simplerace domain. These include :

1. Greedy Controller,
2. Heuristic Sensible and Heuristic Combined Controllers,
3. An evolved multi layer perceptron based controller,
4. An evolved Recurrent multi layer perceptron based controller.

The Greedy controller uses a simple greedy strategy to decide the next move (forward-left or forward-right). Hence it continuously accelerates and tends to overshoot way-points. The Heuristic Sensible controller has a similar strategy but it has a speed limit. If the speed limit is exceeded, it shifts into neutral mode (no acceleration). The heuristic combined controller is more complex and makes strategic decisions using an inbuilt mechanism. It has two modes; In the normal mode, it travels like the Heuristic Sensible Controller, but if the opponent is closer to a waypoint, it enters underdog mode and steers towards next waypoint. If it gets close to the waypoint in underdog mode, it decreases its speed proportionally based on the distance to the waypoint. This is a very clever strategy and serves as a good opponent. The evolved multi layer perceptron based controller (developed by Julian Togelius) is a fairly developed controller that possesses the basic skills required for driving. These controllers can be as opponents for evolving new controllers. It is hoped that Neuroevolution can discover such strategies on its own.

The simplerace package has the required functionality to collect statistics for races between two controllers and also solo races. It also has a CompetitionScore functionality which gives the average of three scores  the solo scores, score against the Heuristic Sensible Controller, score against the Heuristic Combined Controller (each score in turn being the average score obtained in five hundred races).

### 2.3.3 Challenges of Simplerace Domain

A driver should be able to accomplish the basic task of navigating to the current waypoint for which the basic skills of turning, accelerating, braking must be learned. Apart from the basic skills, the simplerace domain presents a lot of scope for innovation and strategy. The following is a list of skills and strategies possible in the simplerace domain

1. Avoid overshooting - While reaching the waypoint, it is important to avoid overshooting. Going too fast can result in missing the waypoint. Or, the waypoint may be captured, but because of the high speed, the car continues travelling in the same direction for an extra distance before readjusting to the new current waypoint. This is called overshoot and it can reduce the number of waypoints captured significantly. To prevent overshoot, it is important to slow down while nearing

the waypoint.

2. Reach the current waypoint in such a way that the next waypoint can be reached quickly - If at the moment of hitting the current waypoint, the car is already oriented towards the next waypoint, the car can efficiently capture the current waypoint and head to the next waypoint. This avoids the time taken for re-orientation and increases the effectiveness of the controller.

3. Overtake the opponent - It is important to be able to overtake the opponent. This may not be possible if the opponent is travelling at the highest possible speed, but an opponent that always travels at such a high speed is prone to overshooting. Good overtaking skills can help 'steal' waypoints from the opponent.

4. Yield to the opponent - Yielding to an opponent is as important as the ability to overtake the opponent. It is a strategic behavior. Realizing the futility of chasing down a waypoint that the opponent is sure to capture can save valuable time. This time can be used to gain an advantage by heading to the next waypoint. Once the current waypoint is captured by the opponent, the controller can easily capture the next waypoint because of the headstart.

5. Use collisions to ones advantage - In simplerace domain, collisions do not cause any damage to the car. Since there is no notion of damage or wear and tear, bumping the opponent controller out of the way can be helpful while chasing waypoints. If the controller is really sophisticated, it can use collisions to exchange momentum with the opponent (collisions in the simplerace domain are elastic).

The goal of this thesis is to evolve controllers that possess such skills and strategies. In the following chapters, the methods used to tackle the car-racing problem in the simplerace domain are explained in detail. In order to put things in perspective, Section 7.2 discusses other approaches that have been successfully used in the simplerace domain.

# Chapter 3

# Direct Evolution

The first approach used to develop controllers for the car-racing problem is Direct Evolution. Direct Evolution is the simplest approach to evolution. It is just a standard implementation of the NEAT algorithm. In the following chapters, three other evolutionary methods are described which have more levels of complexity than the direct approach.

## 3.1 Direct Evolution

For the simplerace car-racing domain, which is a new domain, the best way to learn is to experiment and learn by trial and error. A controller can learn about the domain only by trying out various actions and receiving feedback from the environment. In the car-racing domain, this paradigm of learning by experimentation translates into driving solo. The goal of this approach is to set up evolution, such that the controller learns the basic driving skills and more importantly, learns to capture waypoints efficiently. The hope is that evolution is able to discover some of the advanced skills mentioned in Section 2.3.3 in order to capture waypoints efficiently.

Direct Evolution is a straight-forward implementation of the standard NEAT algorithm. The task used for evaluation is a simple solo race. Each network in the population is evaluated in the simplerace domain and a fitness is assigned. The evaluation stage is followed by a reproduction stage, where the next population is constructed from the current population using the reproduction mechanism of NEAT (Section 2.1). The two stages are repeated until a solution is obtained (or for a fixed number of iterations).

## 3.2 Experiment Setup

The goal of this experiment is to discover driving skills by driving solo races. The track used for racing is a random track (BasicTrack from simplerace package), i.e. the waypoints are created at random. Only the current waypoint and the next way point are known to the controller. Due to the use of a random track, the controller should learn how to drive towards the target waypoints rather than learning a particular track.

The simplerace domain provides relevant information about the first player in an egocentric fashion. The information includes its speed, distances to both waypoints, its angle to both waypoints, etc. In order to drive solo, this information is sufficient. There is however a discontinuity in the domain because of the way angles are measured. A small change in position of the car (when the waypoint is behind the car) can result in the angle to a waypoint changing from $\pi$ to $-\pi$ (or vice versa). To overcome this big jump, each angle is represented as a (sine, cosine) pair which eliminates the jump that occurs at the boundary. Hence the input representation consists of the following seven inputs:-

- speed of the controller,
- distances to both waypoints,
- (sine,cosine) of angle to first waypoint,
- (sine,cosine) of angle to second waypoint.

The controllers have two outputs which are used to control the acceleration/brake and steering respectively. The track used for evaluation is the BasicTrack from simplerace domain. The waypoints are randomly distributed and appear one at a time. They can only be captured in the order of appearance. At any instant of time, the information of the currently active waypoint and the next waypoint is known.

Evolution was carried out for 100 epochs with a population of 200 networks. The fitness was the average number of waypoints captured by the controller in five races, with each race lasting 1000 time steps.

## 3.3 Results

The experiment monitored the progress of evolution by tracking the waypoints captured by the best individual (peak-fitness) and the waypoints captured on an average by the entire population (average fitness). Figure 3.1 shows the Fitness plot (values reported are the average values from ten runs). As seen in the peak fitness curve, evolution is able to discover the skills required to drive solo quite early. By 25 epochs, the peak fitness curve starts to stagnate; The average fitness curve shows reasonable progress up to 40 epochs after which no significant increase is observed.

Table 3.1 shows a comparison of the solo scores obtained by NEAT based controller to scores obtained by the controllers from the simplerace pack. The best controller evolved using direct evolution achieved a score of 19.6 which is significantly better compared to the controllers provided as a part of the domain (Student's t-test,$p < 0.01$).

In addition to achieving creditable scores, Direct Evolution was able to discover some advanced skills. A surprising fact is that all the controllers evolved learned to drive in the backward direction. The actions that the controllers use pre-

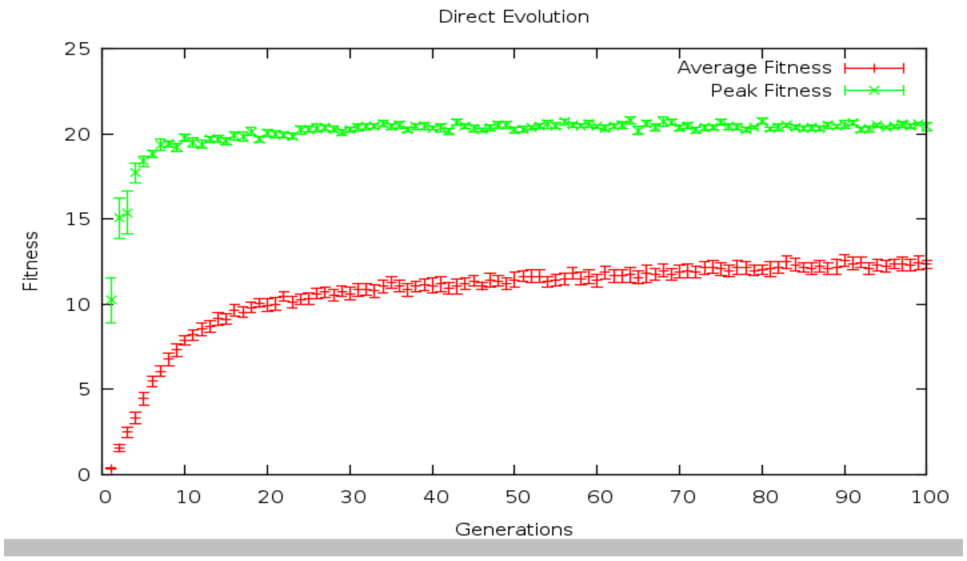

Figure 3.1: Fitness Plot for Direct Evolution.The average and peak fitness at each generation is shown for the duration of the Evolution. Fitness is the average number of waypoints captured by the controller. The peak fitness reaches high values quickly indicating that the basic skills are learned quite early in the evolution. Also no noticeable improvement can be seen after 40 epochs, indicating that the population stagnates.

| Controller | Minimum | Maximum | Average | Methodology |
|---|---|---|---|---|
| Direct Evolution | 1 | 25 | 19.6 | NEAT |
| Heuristic Sensible | 0 | 22 | 13.706 | Hand-coded with domain knowledge |
| Heuristic Combined | 0 | 20 | 7.01 | Hand-coded with domain knowledge |
| Evolved MLP | 0 | 25 | 16.578 | Evolved Multi Layer Perceptron |

Table 3.1: Comparison of Solo Scores. Scores obtained by the NEAT based controller are comparable to controllers provided as a part of the simplerace domain. The heuristic sensible controller, heuristic combined controller were hand-coded controllers and the evolved MLP controller was an evolved recurrent neural network. The NEAT based Controller gets an average score of 19.6 showing that Direct Evolution is capable of evolving skills needed for solo racing

dominantly are back, back-left and back-right. They do not drive forward! This was found to be a common trait across most approaches, confirming something about the domain - i.e. it is harder to control the car when driving forward because forward acceleration is much higher than backward acceleration. Two significantly different and advanced behaviors were evolved using this approach.

1. Aligning towards the shortest distance path :- The controllers learned to adjust their direction such that, it is aligned with the line joining the two waypoints. The straight line joining the two waypoints is obviously the shortest distance between the two waypoints. What this means is that after reaching a waypoint, not too much work is needed to reach the next one, since the controller is already aligned along the shortest path from current position to target waypoint. Also, aligning towards the shortest waypoint avoids overshooting to an extent.

2. V-Turns to save time :- As indicated earlier, most of the evolved controllers drive in the backwards direction. But few of the controllers evolved the ability to switch directions occasionally. When a controller has to completely change directions to go to next waypoint (from a current waypoint), it has to make a complete U-turn. Evolved controllers learned to avoid the U-turn by reversing direction and turning to one side (since the controller normally drives backward, this reversal corresponds to driving forward-left or forward-right in the domain). After this brief reversal and re-orientation, it switches to the old (dominant) pattern of driving backwards. The controller ends up making a clever 'V' rather than a U-turn (Figure 3.2).

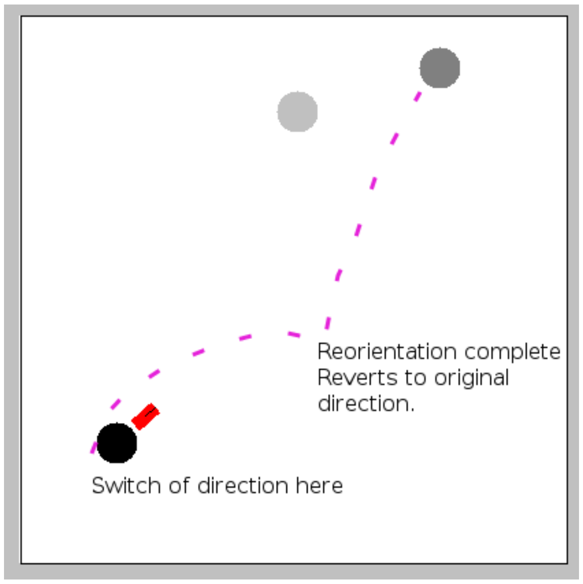

Figure 3.2: Advanced Skill Discovered by Direct Evolution. Figure shows the path traced by controller as it makes a v-turn. The v-turn is faster than a u-turn for changing directions and hence makes the controller more efficient in capturing waypoints.

## 3.4 Discussion

Moving towards randomly distributed target waypoints seems to be a very simple task at the outset but capturing the maximum number of such randomly distributed waypoints in a fixed time-interval is not. Unless one is an expert driver, there are several possible skills like the V-turn discovered by evolution. Experiments reveal that learning by driving solo is not a bad way to develop a controller. This is reasonably intuitive because the overall fitness (i.e. the waypoints crossed in a fixed time) improves directly when the controller learns tricks to avoid overshooting/overturning, and accelerates along straights to the maximum extent.

For the simplerace domain, the controllers evolved using Direct Evolution are very good at capturing waypoints when driving solo. Evolution has discovered the basic driving skills, i.e. accelerate, brake, steering and navigating towards waypoints. The dominant behavior discovered by evolution was to drive in the backward direction. This is because the car becomes harder to control when driving at high speeds due to skidding and larger turning radius. Apart from basic skills, evolution was also able to discover some advanced skills like aligning towards the shortest distance path and V-turns.

The Direct Evolution approach clearly works in the simplerace domain. An important reason is that the fitness measure, i.e. the number of waypoints captured is a highly objective and direct measure for the task at hand i.e. capturing waypoints efficiently. In some cases, such a fitness measure may not be possible. If the goal is to discover high level behavior while racing with opponents (Chapter 5), the number of waypoints captured becomes an indirect fitness measure.

In complex domains with more actions and percepts, it becomes harder to create objective and direct fitness functions that reward good skills and behavior. Instead of evolving the basic skills, evolution can end up finding solutions that exploit some eccentricities of the domain. For example, consider a slight variation of the car racing domain where a solo race (no opponents) takes place on a fixed track and the race takes place for a number of laps. TORCS (Section 7.3) is an example of such a domain. Total distance travelled is one good fitness criterion and lap timing is another. But none of these are fool-proof. If the track is hard enough, none of the controllers produced by evolution will be able to complete a lap and that discourages the use of lap timing as a fitness metric. Distance travelled can also be a tricky metric. Travelling in circles, travelling outside the track, travelling in the wrong direction, etc. have to be taken into account while computing the distance. Depending on the environment, this information may or may not be available. In such situations when there is no obvious fitness metric, Direct Evolution may not work. Other approaches that decompose the task into simpler subtasks may be

| Opponent | Average Score | Average Score of Opponent |
|---|---|---|
| Heuristic Sensible Controller | 10.7 | 11 |
| Heuristic Combined Controller | 13.66 | 14.86 |
| Evolved MLP Controller | 10.22 | 11.69 |

Table 3.2: Direct Evolution controllers lose with opponents. Scores obtained indicate that skills learned from Direct Evolution are not sufficient to race against opponents. The heuristic sensible controller, heuristic combined controller were hand-coded controllers and the evolved MLP controller was an evolved recurrent neural network.

necessary.

Though Direct Evolution was able to learn the basic and some advanced driving skills, there are some skills that were not learned. When directly evolved controllers are raced against opponents, the number of waypoints captured by the controllers drop substantially. Table 3.2 shows the scores obtained by the champion controllers from Direct Evolution when raced against hand coded controllers from the simplerace domain. The scores obtained by the directly evolved controller and the heuristic combined controller and evolved MLP Controller are significantly different (Student's t-test, $p < 0.01$). Against the heuristic sensible controller, the scores are not significantly different, but the average fitness of the champion from Direct Evolution is 10.6 which is almost an eight point drop in fitness. The skills learned while driving solo are not sufficient for driving with an opponent in the simplerace domain. The next chapter addresses this issue by setting up the car racing problem as a series of progressively harder tasks.

# Chapter 4

# Incremental Evolution

Direct Evolution solved the problem of driving solo. The next problem is to learn to drive with an opponent. Driving with an opponent is obviously harder than driving alone. Incremental approaches lend themselves naturally to such problems of increasing complexity.

## 4.1 Need for Incremental Approaches

Trying to solve complex problems head on can be hard for evolutionary approaches. If the task is too demanding, evolution can get stuck in an unfruitful region of the solution space.This is where Incremental Approaches can be useful. Incremental Evolution sets up evolution as a multi-stage process. Each stage is used to solve a slightly different task. The stages are ordered in the increasing order of task complexity. Early stages focus on simple tasks and are used for developing skills/abilities which are essential for later stages. By the time evolution gets to a particular stage, individuals are prepared for the corresponding task. In other words, finding the solution for simple tasks related to the complex problem can help discover a region of the solution space where the complex problem is more accessible.

Further, in the simplerace domain, incremental approach has a clear analogy to the real world. Human drivers do not get into a formula-one race or an inter-state highway to learn driving. They learn the basics of driving on less crowded neighborhood roads. Once they have done that, they practise driving with other-cars on the road. Only drivers who are competent on a variety of roads and cope with traffic can hope to succeed in a car-race like formula-one.

Incremental approaches can be used to evolve complex adaptive behavior. They have been used in improving Non-player characteristic (NPC) behavior in video games([3]) and in improving capture behavior in the well known prey capture domain([5]). They have also been used in the simplerace domain[29]; Incremental Evolution was set up on a series of hand-designed fixed waypoint tracks of varying shapes to master different driving skills and reduce the noise due to random distribution of waypoints. Each Track was designed to master a particular navigational skill (accelerate, brake, turn, reverse, accelerate on long straights, etc.). In this work, incremental evolution is set up to learn to drive with an opponent. Incremental Evolution is set up as a sequence of two tasks : driving alone and racing with an opponent. A separate stage is used for solving each of the tasks.

## 4.2 Experiment Setup

The simplerace domain provides relevant information about the first player in an egocentric fashion (first-person sensors); This includes its speed, distances to both waypoints, its angle to both waypoints, etc. Among the first person sensors, opponent information is specified only in terms of its distances and angle to the first player.

However, to make high level decisions, the controller needs more information about the opponent like the opponent's distance and angle to the waypoints. This information is not directly available, but third-person sensors from the domain can provide this information. Third-person sensors includes position, velocity, orientation of both players and the absolute position of the waypoints. From this the speed of the opponent, and the opponent's angle and distance to both waypoints can be extracted.

The controller has 15 inputs and two outputs. The inputs include:

- controller information - speed, distance to both waypoints, angles (sine, cosine) to both waypoints,
- opponent information - speed, distance to both waypoints, angles (sine, cosine) to both waypoints,
- Explicit binary indicator to indicate the presence of an opponent.

All angles are represented as a sine-cosine pair as described in Section 3.2. The two outputs are used to control the acceleration/brake and steering respectively.

Incremental Evolution is set up as two stages:

- Solo Evolution: The first stage is the same as the previous approach i.e. Direct Evolution where the fitness is determined by the number of waypoints crossed while driving solo. The fitness is the average number of waypoints captured from five trials, each trial being a race of 1000 timesteps. This stage lasts for 100 epochs.
- Driving with an Opponent: Initial experiments with the Incremental Evolution showed that, when solo evolution stage was followed by Driving with an Opponent stage, evolution learned how to drive with an opponent; but the solo driving abilities became impaired. To avoid forgetting previously learned driving skills, 'Refreshing' is introduced into the second stage. In addition to being evaluated on the new task, controllers are also refreshed i.e. evaluated on the old task of driving solo. The fitness of an individual in the second stage is a weighted combination of the score obtained in the two tasks. A higher

weightage is given to the waypoints captured while driving with an opponent because the solo driving task has been learned already and driving with an opponent is the harder task. In both the tasks, the fitness is the average number of waypoints captured in five trials, each trial being a race of 1000 timesteps. The second stage lasts for a total of 200 epochs which is divided into two equal sub-stages. For the first hundred epochs, the simple heuristic sensible controller from simplerace domain is used as the opponent. For the remaining 100 epochs, the slightly more advanced heuristic combined controller is used as the opponent. This sequence ensures that even during the second stage, the task becomes increasingly hard and encourages evolutionary progress.

The track used (BasicTrack) and the population size (200) are the same as those used in Direct Evolution.

## 4.3 Results

Figure 4.1 shows the progress of Incremental evolution through 300 generations. The results have been averaged over ten runs. While direct evolution (Figure 3.1) stagnated after 40 epochs, Incremental evolution progress much longer (until 150 epochs). However, it is not able to maintain progress throughout the second stage. The second stage lasts for 200 epochs, and out of these 200 epochs, progress is maintained only for the first 50 epochs. This is a cause of concern and the topic is revisited in Section 4.4.

Tables 4.1 and 4.2 show how Incremental Evolution helped improve racing with different opponents. The comparisons are based on the ten best controllers (since there were ten trials) evolved from each approach. Each champion controller was evaluated with the opponent for 500 races of 1000 timesteps each and the average

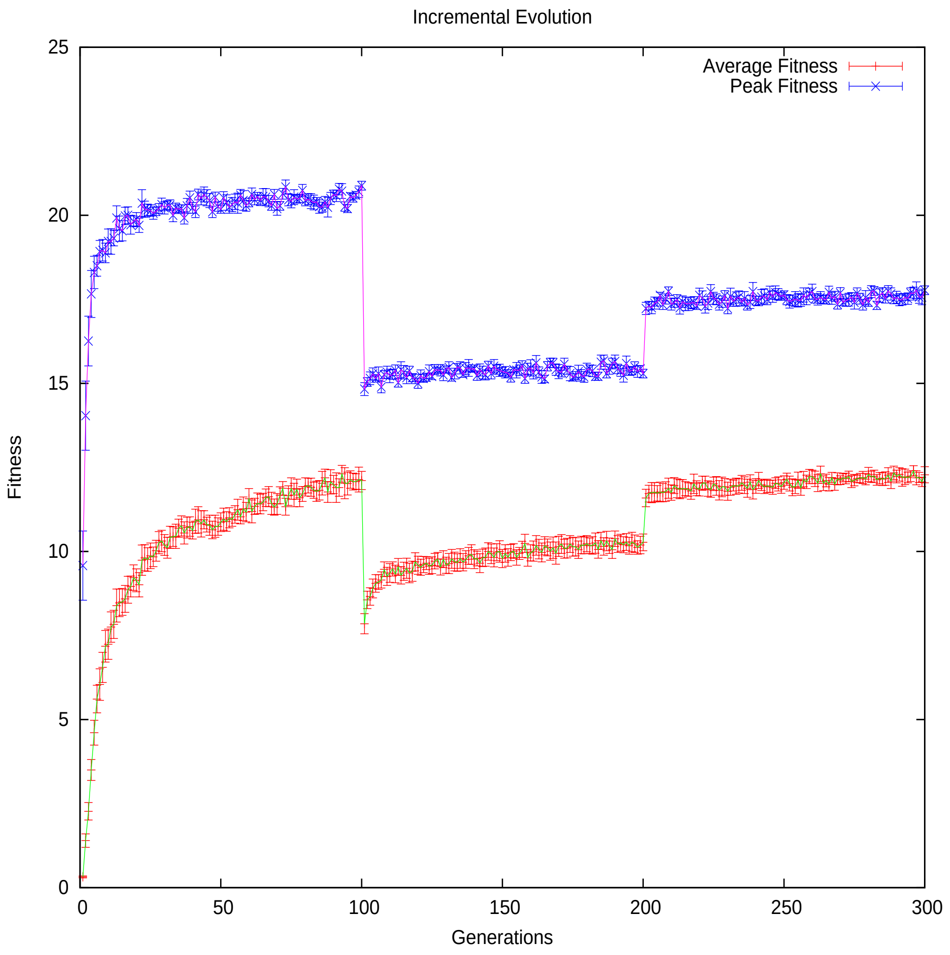

Figure 4.1: Fitness Plot for Incremental Evolution. The average and peak fitness at each generation is shown for the duration of the Evolution. Incremental Evolution progresses longer than Direct Evolution and hence can discover better solutions

| Opponent | Direct Evolution Champion | Opponent Score | Incremental Champion Score | Opponent Score |
|---|---|---|---|---|
| Sensible | 10.6 | 11.14 | 12.28 | 10.8 |
| Combined | 13.66 | 14.86 | 15.1 | 14.96 |
| Evolved MLP | 10.22 | 11.69 | 11.5 | 10.92 |
| Fixed Competitor | 11.96 | 8.7 | 12.52 | 8.75 |

Table 4.1: Waypoints Captured By Incremental Approach and Direct Evolution. Incremental Approach produces controllers that can win against opponents

| Opponent | Direct Evolution | Incremental champions | Difference in Victory Margins |
|---|---|---|---|
| Sensible | -0.54 | 1.48 | 2.02 |
| Combined | -1.2 | 0.15 | 1.35 |
| Evolved MLP | -1.47 | 0.58 | 2.05 |
| Fixed Competitor | 3.25 | 3.77 | 0.45 |

Table 4.2: Victory Margins for Incremental and Direct Evolution. The margin of victory is better for Incremental Controllers as the controllers produced by Incremental Evolution are more skilled in racing with opponents

waypoints captured were recorded. The numbers presented here are the averages of the respective controllers from ten trials.

When racing with an opponent, following factors are considered important:

1. Number of waypoints captured: While racing against opponents like the evolved MLP controller, Incremental Evolution is found to capture more waypoints than Direct Evolution (Student's t-test, $p < 0.05$).

2. The margin of victory: For example, with the Evolved MLP Controller (an evolved non-recurrent MLP provided with the simplerace domain), the controllers developed using Direct Evolution ended up losing by a margin of 1.47 waypoints on an average. On the other hand, controllers developed using

> Incremental Evolution ended up winning the races with the MLP controller and had a positive margin of 0.58. The margins of victory for incrementally evolved controllers are significantly better (Student's t-test, $p < 0.01$).

Effectively, incrementally developed controllers captured two waypoints more in every race than directly evolved controllers (except while racing with the fixed competitor which was an easy opponent). Further, a one-on-one race between the best controllers developed using both the approaches also showed the incremental evolved controller capturing two waypoints more than the Directly evolved controller. Using a two-stage process, Incremental Evolution was able to produce controllers capable of defeating opponents.

## 4.4 Discussion

As shown in Chapter 3, controllers developed using Direct Evolution were not able to race effectively with opponents. To tackle this problem, evolution was set up as a sequence of of progressively harder stages. The first stage was the same as direct evolution and the goal was to learn driving solo. The goal of the second stage was to learn to drive with an opponent. In order to ensure that controllers did not forget previously learned skills, the second stage involved a refresh component. The fitness function for the second stage was the weighted sum of scores from solo races and races with an opponent. During the second stage, two different opponents were used to encourage the progress of evolution.

Using such a setup, Incremental Evolution was able to produce controllers capable of driving with an opponent. On an average, the incrementally evolved controllers captured two waypoints more than directly evolved controllers in a race of 1000 timesteps while racing with the same opponent. However, this approach was not able to discover new skills and strategy. Though the evolved controllers can race with opponents, they do not display any of the high-level strategy described in Section 2.3.3.

Figure 4.1 shows that progress lasts only for the first 50 epochs of the second stage (200 epochs total). For the rest of the second stage, including racing with the second opponent (Heuristic Combined Controller), the fitness does not change significantly. This stagnation can prevent evolution from discovering more innovative controllers. A possible solution is to hand design more opponents and increase the number of sub-stages in the second stage. Though having a pedagogy of increasingly tough opponents is helpful, hand designing such opponents is a challenge in itself.

The need of the hour is to ensure that the population's fitness does not stagnate, for which the opponents must keep improving in each generation. It is impossible to manually design one opponent for each generation of evolution. In the next chapter, an evolutionary approach is introduced that, 1)Provides candidate opponents automatically 2)Ensures continuous improvement of opponents by evolving them.

# Chapter 5

# Competitive Coevolution

The last chapter illustrated an incremental approach to solving the problem of racing with opponents. The main problem with this approach is that the opponent is fixed. If the fixed opponent is fairly advanced, the individuals from early generations of evolution end up getting very low fitness and evolution never takes off. On the other hand, if the fixed opponent is too simple, apart from learning to exploit weaknesses of the opponent(overfitting), it prevents evolution from improving continually and yielding better solutions. Once evolution discovers a good strategy, evolution can stagnate. One way to avoid this is to evolve against a sequence of opponents starting with a simple opponent and gradually increasing the complexity and skill level of the opponent. The approach of evolving against a sequence of opponents will only work if such opponents are automatically available which is normally not the case. This chapter discusses a coevolutionary approach to overcome this problem by evolving opponents as part of the process.

## 5.1 Coevolution and Competitive Coevolution

Coevolution is defined as a 'process of mutual adaptation that occurs amongst a set of agents that interact strategically in some domain'[4]. By definition, a coevolutionary domain is interactive. The key difference between Evolutionary algorithms and Coevolutionary algorithms lies in the evaluation. In Coevolution, the fitness of the organisms depend on one another. Competitive coevolution is the classical case of coevolution in which individual fitness is evaluated through competition with other individuals in the population, rather than through an absolute fitness measure. Normally, there are two populations(though coevolution can be implemented with single or multiple populations); the population currently being evaluated is called the 'host' and the population from which opponents are drawn is called the 'parasite' population. During each evaluation an organism from the host population is evaluated against an organism from the parasite population. The goal of the host is to overcome the parasite and the goal of the parasite is to exploit the weaknesses in the host. One of the advantages of Competitive Coevolution is that a precise fitness function is not needed for the domain, a measure of the relative strengths of solutions is sufficient. An increased fitness in one solution leads to a decreased fitness for another. Further opponent strategies need not be designed by hand. As coevolution progresses, competing solutions strive to improve in order to survive. This can lead to an 'arms race' of increasingly better solutions.

Competitive coevolution flourishes as long as the arms race continues. In practice, it is difficult to ensure that the arms race takes place[4]. This is because coevolution is susceptible to forgetting. Given a finite population, an individual must be successful in every generation in order to survive. Otherwise, it is eliminated from the population and the unique skills(if any) learned by this individual are gone and have to be rediscovered. Hence, skills learned in the past can sometimes be forgotten by the entire population. In conventional evolutionary algorithms, forgetting is not a big problem because the lost individual is strictly worse in terms of an absolute fitness measure and is replaced by better individuals. However, in coevolution the fitness is not absolute and an individual with low fitness may resurface again or an individual with good skills maybe lost.

Several techniques have been proposed for overcoming forgetting. One such technique is to maintain a Coevolutionary Memory(CM) like Hall of Fame[22] or Layered Pareto Coevolution Archive(LAPCA). CM helps overcome forgetting by storing the best players and hence the strategies learned by evolution. While evaluating a host, it is not only evaluated against a parasite but also against individuals picked from the CM. Evaluating against a CM ensures that past skills and strategies are not forgotten.

The Hall Of Fame(HOF) is a best of generation coevolutionary memory where the fittest individual from each generation is stored. A host is evaluated by playing against a parasite and also against individuals from the HOF. It is sufficient to play against a random sample of opponents picked from HOF rather than evaluating against the entire HOF. HOF is used as the CM because it is easy to implement and smaller number of games need to be played.

To monitor progress in Coevolution and select the overall champion, the dominance tournament method is used. Dominance Tournament was proposed by Stanley and Miikkulainen[26] as a method to guarantee that strictly more sophisticated strategies are discovered as evolution progresses.

Dominance is defined recursively as follows:-

- The first dominant strategy $d_1$ is the champion of the first generation.
- Dominant strategy $d_j$, where $j > 1$ is a generation champion such that for all $i < j$, $d_j$ is superior to $d_i$. where the definition of 'superior' depends on the domain.

This definition ensures that there are no dominance cycles as each successive dominant strategy has to be superior to all the previous dominant strategies.

The process of deriving such a ranking from a population is called dominance tournament. The main advantage of this approach compared to other approach like master tournament is that the number of comparisons needed to establish such a ranking is significantly reduced. If the candidate strategy $d_x$ is not superior to the first dominant strategy $d_1$, $d_x$ can be rejected without comparing it with the rest of the dominant strategies.

## 5.2 Experimental Setup

In the simplerace domain, both the player and the opponent have the same task i.e. to capture maximum number of waypoints. The game is symmetrical as player one and player two can be interchanged. Hence it is sufficient to use just one population. Both the host and the parasite are picked from the same population. Consequently, the Hall of Fame is also filled with the Best of Generation individual from this population. In the rest of this chapter, the host is referred to as 'individual' or 'player'; and the parasite is referred to as opponent.

The population size is fixed at 200 and each evolution runs for 100 epochs. The fitness of an individual is the average score obtained from three components. Each component includes a series of races against a set of opponents. The opponents are not fixed but are picked from the evolving population. Each set of opponents have a role to play in competitive evolution. The components and their importance are described below.

1. Number of Waypoints captured while racing with Best of Species Opponents: It is essential to make sure that the individual is evaluated against good opponents. Since NEAT has an explicit speciation mechanism, we can take advantage of this to select a diverse set of good opponents. Like Stanley and Miikkulainen in [28], the champion individual from the four best species are selected to be the opponents. Since opponents are selected from different

species, they are likely to be diverse. Each opponent plays one race of 1000 timesteps with the player. The fitness awarded to the player is the average number of waypoints captured with the four best of species opponents.

2. Number of Waypoints captured while racing with HOF opponents: Racing with HOF opponents is done to make sure that previously learned skills are not forgotten. Four opponents are picked at random from the Hall of Fame. Again, each opponent plays one race of 1000 timesteps with the player. The fitness awarded to the player is the average number of waypoints captured with the four opponents.

3. Number of Waypoints captured while racing with Random Opponents: The motivation for playing with random opponents is to make sure that the player learns to race with different kinds of opponents. Evaluating only with HOF opponents or the Best of Species opponents may help the player play against strong opponents, but it could also lead to a player overfitting a few good opponent strategies. Each opponent plays one race of 1000 timesteps with the player. The fitness awarded to the player is the average number of waypoints captured with the five opponents.

At the end of each generation, the best individual of the generation is added to the HOF. In the dominance tournament, the comparison between two strategies is based on five races of 1000 timesteps each. That is, an individual $x$ is superior to individual $y$ if $x$ captures more waypoints in aggregate than $y$ over the five races.

The input representation is the same as the input representation used in the incremental approach. The controller has 15 inputs and two outputs. The inputs include:

| Controller | Margin of Victory |
|---|---|
| Direct Evolution | -1.55 |
| Incremental Evolution | 3.48 |
| Coevolution | 1.33 |

Table 5.1: Comparison of Margin of Victory for Direct, Incremental and Coevolutionary Approaches. Coevolved controllers have a higher margin of victory than Direct Evolution controllers when racing with opponents. This shows that coevolution is capable of discovering how to race with opponents

- controller information - speed, distance to both waypoints, angles(sine, cosine) to both waypoints,
- opponent information - speed, distance to both waypoints, angles(sine, cosine) to both waypoints,
- Explicit binary indicator to indicate the presence of an opponent.

The two outputs are used to control the acceleration/brake and steering respectively.

## 5.3 Results

Coevolved controllers can race with opponents as well as drive solo. The ability to race with opponents is natural because the controllers evolve by competing against other controllers in the population. Unlike the incremental approach, which required multiple stages(300 epochs) for learning the two tasks, coevolution is able to gain both sets of skills within 100 epochs. The following table(Table 5.1) shows a comparison between the best controller evolved using the three approaches. The Margin refers to the aggregate margin of victory(in terms of waypoints) while racing with three good opponents - i.e. the simplerace heuristic controller, the simplerace combined controller, and the evolved MLP controller provided in the domain.

The margins for the three approaches were significantly different (Student's t-test,$p < 0.05$). The coevolved controllers are clearly better than the controllers evolved using Direct Evolution. However, the incrementally evolved controllers have the best margin. The high margins could be due to the fact that two out of these three opponents(heuristic sensible and heuristic combined controllers) were used during incremental evolution as opponents and hence the incrementally evolved controllers had experience racing against them.

The main goal of this approach was to discover innovative behaviors and the controllers evolved some new behaviors:

1. V-turn: The directly evolved controller had learned to switch directions, and made a nice V-turn(Section 3.3). This was however a short-lived behavior as the controller reverted to its original backward driving mode after making the adjustment in orientation. Coevolved controllers were also able to discover this behavior.

2. Forward and Backward Driving: In previous approaches, most of the evolved controllers preferred to drive backward. Coevolved controllers were able to travel in both directions. Typically, coevolved controllers drive much faster when they are driving forward. When they drive backward, they are slower but much more refined. Ideally one would expect these controllers to slow down when they near the waypoint, but while travelling forward they occasionally accelerate too much and overshoot the waypoint. If this overshooting is eliminated, these controllers can capture much more waypoints while racing with opponents. In spite of the overshooting, the discovery of the forward and backward-driving behavior represents a progress from the earlier approaches.

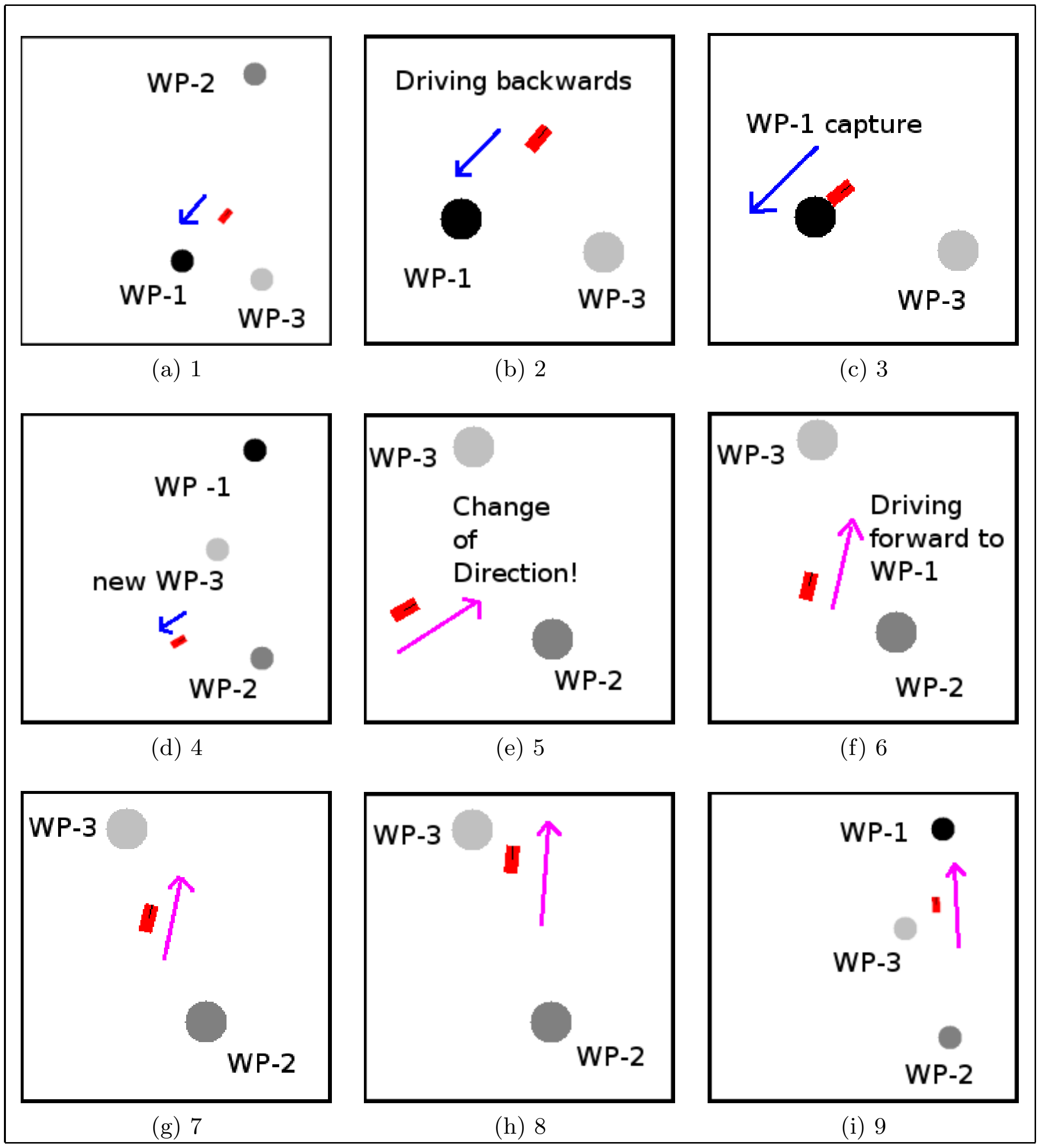

Table 5.2: Forward and Backward Driving. A sequence of nine screenshots from a solo race of a coevolved controller is shown from top left to bottom right. The first, fourth and ninth screenshots show the entire 400x400 field; the rest of the screenshots show only the part of the field containing the action. The reversal in direction can be seen in the fifth image. This ability to drive in both directions for long distances gives the controller an edge in capturing waypoints. Note: The little black line on the red-car that indicates the front of the car. WP-1, WP-2 represent the first and next waypoints respectively. WP-3 is not a part of the sensor model and is shown only for visual cue.

## 5.4 Discussion

The coevolution approach was set up to encourage continuous progress during evolution. Dominance tournament was used to monitor progress and HOF was used as the coevolutionary memory. Controllers evolved using this approach were able to race solo and drive with an opponent. Though the coevolved controllers were better than directly evolved controller, incrementally evolved controllers had the highest margin of victory. In spite of this fact, coevolved controllers possessed a few interesting skills. Developing skills solves only one part of the problem. The second goal of discovering strategies was not accomplished by coevolution. In the next chapter, an approach is outlined that explicitly encourages the discovery of high level behavior by problem decomposition.

# Chapter 6

# Modular Evolution

Competitive Coevolution enabled the controllers to learn the basics of driving and a few impressive skills, but no high-level strategy emerged. Although the simplerace domain provides opportunities for strategy (Chapter 2), evolution was not able to discover them on its own. This chapter proposes another approach to evolution - a modular approach, where the problem is decomposed into smaller sub-problems.

## 6.1 Modularity

Modularly constructed systems are necessary to solve complex real-world problems, because they allow subproblems to be isolated and solved separately[16]. Two concepts are central to modularity: Decomposition and Replication. In Nature, one can find numerous examples of replication: limbs, eyes, ears, etc. are replicated and can be considered as modules which perform the same function. Indeed there is a strong biological motivation to develop artificial systems that have symmetry. In Computer Science too, the divide and conquer paradigm encapsulates the notion of decomposing a complex problem into smaller sub-problems which can be solved easily. In neural network literature, Modular Neural Networks have been evolved for solving problems like time series prediction[16] and robot control([30],[17]).

Instead of trying to find a single solution which has all the skills and strategies needed to be an intelligent driver, dividing the car-racing problem into inde-

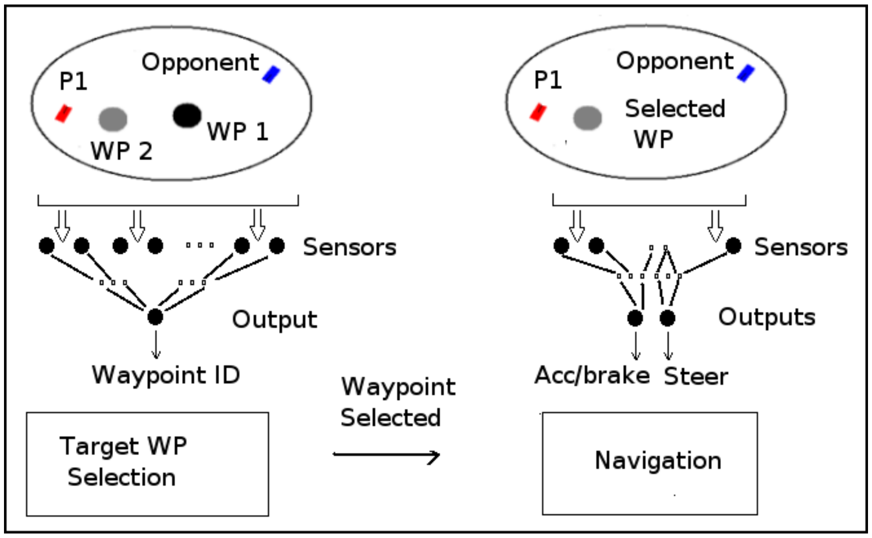

Figure 6.1: Modular Design. The Waypoint selection and Navigation modules are shown. P1 represents Player 1; WP-1 and WP-2 are the current and next waypoints respectively. The waypoint selection module selects the waypoint, which is passed to the navigation module. Car-racing is decomposed into two separate modules as shown with a goal of evolving strategy

pendent tasks makes it easier for evolution to discover driving skills and interesting strategies.Car-racing in simplerace domain is decomposed into 2 tasks : Navigation and Waypoint Selection. The Waypoint selection module involves deciding a target waypoint. It takes into account the controller's current state (position, velocity) and the opponent's current state and the position of the waypoints to select a target waypoint. Once a target is selected, the role of the navigator is to travel to that waypoint in the shortest possible time. Waypoint Selection and Navigation are two separate tasks and by isolating the tasks, it should be easier to evolve 1)Intelligent waypoint selection and 2)Improved navigation. Improved waypoint selection is a high-level strategic decision. By designing modules in the above fashion, evolution is provided with a structure to learn high-level decision making.

The inputs to the navigator include the car's current speed, the car's distance and angle to the target waypoint, the opponent's speed, and the opponent's distance and angle to the target waypoint. The navigator has two outputs: one to control steering and one for driving. The waypoint selector has 15 inputs and one output. It takes into account the speed, distances to both waypoints, and angles to both waypoints of both the controller and the opponent. The single output is used to decide the target waypoint.

If there is no opponent (this information is explicitly given as a binary input to both navigator and waypoint selector), all information regarding opponent is zeroed out. If there is no opponent, there is no role for target selection, the controller should always drive towards the first waypoint. Recall from Chapter 2 that the second waypoint becomes available for capture only after the first waypoint has been captured. So, evolution has to figure out this subtle difference between driving alone and driving with an opponent.

## 6.2 Experimental Setup

In this approach, the navigator and waypoint selector controllers are evolved together by having two populations: one each for the navigator and waypoint selector. The populations consist of 200 networks each and the evolution is carried out for 100 epochs.

For the purpose of evaluation, an organism from the waypoint selector population is paired up with five different navigators from the navigator population. Each such pair is assigned a fitness that is a weighted combination of the average number of waypoints captured during 1)Five races of driving alone and 2)Five races of driving with a fixed opponent. Because it is harder to drive with an opponent, the waypoints captured while racing with an opponent are weighted higher than waypoints captured while driving alone. Each race is set up to last 1000 timesteps.

The fitness assigned to the networks (both navigator and waypoint selector) is the best fitness obtained from all their pairings. The opponent used for this approach was the simple heuristic controller from the simplerace domain (Chapter 2).

## 6.3 Results

Experiments show that the champion controller produced by using the Modular approach is better than the controllers produced by the Direct Evolution approach for racing with an opponent. However, incremental and coevolutionary approaches achieve slightly better scores while racing with an opponent. Direct Evolution is found to be better than all the other approaches for evolving the skills needed to drive solo. The main advantage of modular approach is that the approach is able to evolve clearly discernible high-level strategy.

This approach was designed to help evolution discover solutions with the ability to make intelligent high-level decisions (selecting waypoints) and also to navigate effectively. Further evolution had to realize that the waypoint selection is trivial while driving solo. Controllers evolved using this approach were able to achieve all the above goals. Waypoint selectors were able to make the (trivial) choice of always navigating to the first waypoint while driving alone. In addition, evolution was also able to discover two related high-level behaviors - Waypoint Selection and Hang-Around, which are described below.:

1. Waypoint Selection Behavior: This behavior may appear to be a straightforward and easy to accomplish. Assume using the following heuristic: if the opponent is closer to the first waypoint, the controller should travel to the second waypoint, otherwise it should travel to the first waypoint. One such controller (heuristic combined controller that is provided as a part of the

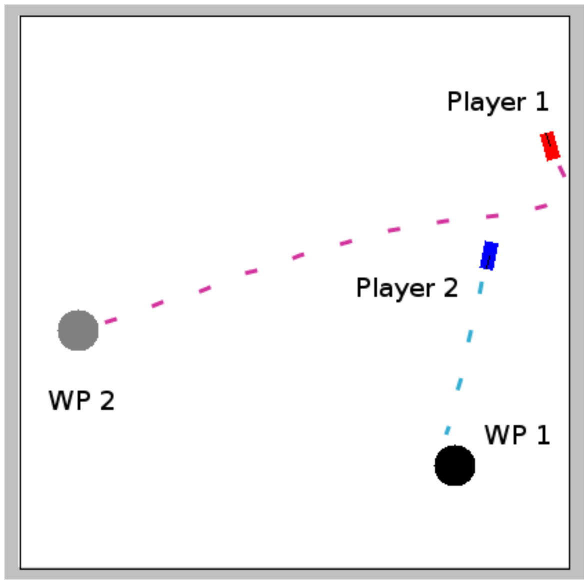

Figure 6.2: Intelligent Waypoint Selection. The figure shows the path traced by modular controller (player 1) when the opponent (player 2) is close to the current waypoint. The controller sacrifices the current waypoint and travels to the next one. Such intelligent behavior can save a lot of time in this car-racing domain. Note that the modular controller travels predominantly in the backward direction.

domain) does reasonably when driving with an opponent but struggles while driving solo (Table :3.1). The waypoint selection decision is not that simple. There are factors like speeds of the two players and the direction in which the two players are headed that should be considered in the decision making. If the player is heading in one direction in high speed, and the next waypoint appears somewhere close but in the opposite direction, the player is going to find it hard to slow down and change directions and get to this waypoint. If the other player uses the above heuristic to make the decision, it is going to miss out on valid scoring opportunities. Also, during the course of the game several other scenarios may arise; one of the players may get stuck near the first waypoint (Chapter 7). If the other player uses the simple distance heuristic and navigates to the second waypoint, the game is stalled because waypoints need to be captured in their order of appearance. Handling every possible scenario that can arise in a domain is not easy and this approach does not scale up. In such cases its advantageous to use evolution to discover strategies by repeatedly playing the games. Figure 6.2 shows the ability of the evolved controller to select waypoints intelligently.

2. Hang-Around: Hang-Around behavior is related to the waypoint selection behavior. In some scenarios, the controller makes an early call and selects the second waypoint. It travels to the second waypoint and reaches it. But the second waypoint cannot be captured until the other player captures the first waypoint. So the controller has to wait - but it cannot afford to lose the advantage gained by moving away from the second waypoint. Interestingly, evolution has discovered an interesting behavior where it tries to capture the second waypoint repeatedly almost as if in a loop. This behavior ensures that, the controller does not tread too far from the second waypoint and hangs

around in the vicinity of the waypoint. Further the controller does not throw away the advantage earned by making a strategic decision to go for the second waypoint. In rare scenarios where the opponent gets stuck trying to capture the current waypoint, the modular controller was observed to reverse its initial decision and get back to the current waypoint. These observations indicate intelligent decisions made dynamically according to the state of the game.

The controllers evolved using the modular approach were able to make intelligent decisions like waypoint selection and hang-around. These intelligent decisions relied on the navigational skills of the controller. This decision making behavior represents a progress over the previous approaches.

## 6.4 Discussion

Evolution of high level strategy for complex games is hard. Neuroevolution methods fail to discover high level strategy in the simplerace domain without any domain knowledge. A modular approach was setup to explicitly encourage evolution of high level strategy. This was done by using domain knowledge and decomposing the car-racing problem into two sub-problems viz. waypoint-selection and navigation. Once such a decomposition was setup, modular evolution was able to discover high level behavior and learn navigational skills. This methodology based on task decomposition and modularity is a promising approach for evolving strategic behavior. In this thesis, the modular approach using Neuroevolution was used in a car-racing simulation, but it can be extended to evolving strategy in other domains and tasks.

# Chapter 7

# Discussion and Future Work

The goal of the thesis was to evolve intelligent behavior for challenging physical simulations, in particular simulated car-racing using Neuroevolution. Towards that goal, four approaches viz. Direct Evolution, Incremental Evolution, Coevolution and Modular Evolution were implemented using the NEAT algorithm. The four approaches were evaluated on the simplerace car-racing domain. In this chapter, the strengths and weaknesses of each of the approaches are analyzed and areas for future work are identified.

## 7.1 Lessons Learned

Direct Evolution was used to evolve controllers by driving solo races and experimenting with different actions. The controllers evolved using this approach were able to learn the basics of driving and also a few advanced driving skills. Only seven inputs were needed to implement this approach :- the first player's speed, its distance to both waypoints, and its angles (represented as a sine, cosine pair) to both waypoints. Advanced skills like aligning towards the shortest distance path and V-turns enabled this controller to get high solo scores. The solo scores of controllers obtained using this approach were comparable to the solo scores of winners of the CIG and CEC 2007 competitions. Out of the four approaches evaluated, the controllers produced by direct evolution were the best in capturing waypoints while driving solo.

However such controllers were not able to race well with opponents. The hand-coded controllers from the simplerace domain were able to defeat the directly evolved controllers. Racing with opponents is harder than driving solo. In order to deal with increasingly hard tasks, an incremental approach was implemented.

The incremental approach was implemented in two main stages; the first stage was solo evolution and the second stage was racing with an opponent. To counter the forgetting of previously learned abilities, a refreshing component was introduced in the second stage. In addition to being tested with the opponents, the second stage also involved a test to refresh the skills learned during the first stage. Controllers evolved using this approach were able to race better with opponents. An additional improvement made was to split the second stage into multiple sub-stages, where each successive sub-stage had a stronger opponent. This approach was found to yield better results than racing with a single opponent. Due to these changes, the controllers evolved using the incremental approach captured two more waypoints per race when compared to the directly evolved controllers. Though the incremental approach was successful in evolving controllers capable of racing with an opponent, no intelligent behavior was observed i.e. no strategies were discovered by evolution.

Competitive Coevolution was implemented to overcome the drawbacks of the incremental approach. Competitive Coevolution usually involves two populations competing against each other. Since the simplerace domain is symmetric with respect to the two players, a single population was sufficient. As the individuals of the population compete with each other and strive to stay ahead of the rest of the population, this approach encourages continual progress during evolution. Controllers evolved using this approach were able to race with an opponent as well as race solo, but they were not the best in either task: they were slightly inferior to incrementally evolved controllers while racing with opponents and to directly evolved controllers while racing solo. The key benefit from this approach was in terms of observed behavior. Coevolved controllers were capable of driving forward and backward and switching directions when needed. The directly evolved controller switched to the forward direction momentarily while doing a V-turn, but this behavior was short-lived and occurred only during the V-turn behavior. But coevolved controllers were able to drive in both forward and reverse directions for significantly long periods of time i.e. turn, accelerate, slow down, etc. This behavior is difficult for human drivers, but coevolution was able to master this skill. However, discovering such advanced skills is only part of the problem. Coevolved controllers were not able to evolve strategy.

The modular approach was designed bottom up to encourage evolution to discover strategic behavior. The car-racing problem was decomposed into two sub-problems viz. Waypoint Selection and Navigation. The Waypoint Selector acted as a high level module and selected the target waypoint. The navigator's role was to navigate effectively to the target waypoint. The Navigator and Waypoint Selector populations were evolved together. Given such an explicit decomposition of tasks, evolution was able to discover solutions to the target waypoint selection and navigation sub-problems. The evolved controllers were able to select the target waypoints intelligently based on the opponent's proximity, speed and orientation to the current waypoint. The controller also evolved the ability to wait for the opponent to capture the current waypoint (if needed) while hanging around near the next waypoint. One drawback was that the target waypoint selection was not perfect. The target waypoint selector module, occasionally made incorrect waypoint choices, this sometimes resulted in the loss of a waypoint. However this did not happen frequently and the advantage gained due to this behavior far outweighed the occasional loss of a waypoint. The modular approach was able to evolve strategic behavior for the car-racing domain. An explicit decomposition of tasks was necessary to achieve high level behaviors using neuroevolution. These experiments demonstrated the use of a modular neuroevolution approach to evolve strategy in car-racing simulations but it can be extended to evolving strategy in other domains and tasks.

## 7.2 Comparative Study

In this work, NEAT has been used in the simplerace car-racing domain to evolve strategies and advanced skills. To put things in perspective, it is important to compare the approaches described in this thesis with other learning methods that have been used in this domain. The CIG 2007 and CEC 2007 car-racing competitions used simplerace domain as the racing platform and hence winners from these competitions can be considered as powerful controllers implemented using other learning methods. This section does two types of comparisons. The first comparison looks at the scores obtained by controllers while racing solo and the second comparison analyzes the scores obtained while racing with an opponent. Before making the comparisons, the top two controllers from each competition are described.

The winning controller from the CIG 2007 competition was developed by Pete Burrow who pioneered modular approaches in the car-racing competition. Burrow's controller used a simple genetic algorithm to evolve two modules called action and decision which were similar to the navigation and waypoint selection modules described in this thesis. However, there were some important differences. Burrow used an ordinary MLP for the decision module and a recurrent MLP for the action module. The topologies for these modules were hand-designed and the weights were evolved for 500 generations using a population of size 30. The decision module was evolved first in solo races, after which the action module was evolved by racing with an opponent. Further, the decision module had only three inputs and made use of domain knowledge: the ratio of speeds of the controllers, and the ratio of distances of the two controllers to the two waypoints. The runner-up controller from the CIG 2007 competition developed by Thomas Haferlach, was based on a modified CoSyNE algorithm[7]. It used two modules, one for driving and one for steering. The neural networks were evolved using a population size of 100 for 2000 generations.

The winning controller from the CEC 2007 competition developed by Ho Duc Thang and Jon Garibaldi, used a heuristic internal controller(which used internal simulation of the race) for target waypoint selection and a Non-stationary fuzzy system for navigation. The runner-up in the competition was developed by Tomoharu Nakashima and his students at Osaka Prefecture University. It used a hybrid model based on controllers from the package and the competition. A combination of two neural networks and the heuristic sensible controller from the simplerace domain were used for low-level decision making (navigation). In addition, four selection rules and an exception handling mechanism constructed using domain knowledge were used to select one of the three controllers. For high-level decision making, a heuristic mechanism was used. The two neural networks used for navigation were developed using a combination of TD learning and Evolutionary computation. Details about the various approaches used in the CEC competition including Genetic Programming and Fuzzy Systems can be obtained in [29].

For the first part of the comparative study, the solo scores of the controllers developed using NEAT are compared with two best controllers from the two car-racing competitions. As the Table 7.1 shows, the NEAT based controller performs statistically better than the CIG 2007 winner (Student's t-test, $p < 0.05$). Though the NEAT controller has better scores than the CEC runner-up and CIG runner-up,

| | | | | |
|---|---|---|---|---|
| NEAT | 1 | 25 | 19.6 | Direct Evolution using NEAT |
| CIG 2007 runner up | 7 | 26 | 19.338 | Modular Controller based on CoSyNE |
| CIG 2007 winner | 0 | 25 | 18.648 | Evolved Modular Controller |
| CEC 2007 winner | 17 | 32 | 24.24 | Non-stationary fuzzy system |
| CEC 2007 runner up | 2 | 27 | 19.354 | Hybrid model using Neural networks |

Table 7.1: Comparison of Solo Scores with Competition Winners. Solo scores represent the average scores obtained from 500 races of 1000 timesteps each. The scores obtained by the NEAT based controller are comparable to winning controllers from the CIG and CEC car-racing competition.

| | | |
|---|---|---|
| NEAT | 15.3 | Modular Evolution using NEAT |
| CIG 2007 runner up | 16.9 | Modular Controller based on CoSyNE |
| CIG 2007 winner | 15.2 | Evolved Modular Controller |
| CEC 2007 winner | 20.3 | Non-stationary fuzzy system |
| CEC 2007 runner up | 19.5 | Hybrid model using Neural networks |

Table 7.2: Comparison of Competition Scores. Competition Score is the average of 3 components: average solo score, average score while racing with Heuristic Sensible Controller and average score while racing with Heuristic Combined Controller. The scores obtained by the winners from the CEC car-racing competition are better than the NEAT based controller. Approaches which use internal simulation and heuristic knowledge performed much better than approaches which did not internally simulate the race.

the scores are not statistically significant. The CEC 2007 winner is significantly better than NEAT based controller (Student's t-test, $p < 0.05$); However, the CEC 2007 winner cannot be used in real time because the controller internally simulates the actual race for selecting the action at each timestep and this makes it extremely slow.

For the second part of the comparative study, Competition Scores are considered. The Competition Score is the average score obtained while racing solo, racing with the heuristic sensible controller and racing with the heuristic combined controller. Table 7.2 shows the Competition Score obtained by the best controllers from the CIG and CEC car-racing competitions. Burrow's evolved modular controller and the NEAT based controller were inferior compared to the other controllers. (Student's t-test, $p < 0.05$). There was no significant difference between NEAT based controller and Burrow's evolved modular controller. One of the important reason for such a huge difference between the winners of the two competitions was that most of the top ranked controllers in the CEC competition used internal simulation of the race and heuristic knowledge to generate the next action.

In summary, the more successful controllers from the CEC competition leveraged internal simulation, heuristic knowledge and hybrid models. Though they were successful in achieving high competition scores, such approaches cannot be directly applied to other physical simulation domains in which humans are not experts.

Out of the controllers discussed in this section, the NEAT based controllers, Burrow's evolved modular controller and Haferlach's CoSyNE based controller were based on Neuroevolution. These three NE methods were the only approaches which 'discovered' strategic capabilities. Haferlach's CoSyNE based approach was the closest to the NEAT based approach used in this thesis because it used minimal domain knowledge to evolve the strategic waypoint selection behavior. Although Burrow's evolved modular controller developed this strategic capability, the input representation used by Burrow leveraged human expertise and understanding of speeds, distances and their ratios. Like NEAT, CoSyNE has been shown to be a powerful NE method for control tasks. Since all the NE methods discovered strategic capabilities, the higher competition score for CoSyNE based approach may mean that CoSyNE is better than NEAT for solving the underlying control task in the simplerace domain. However, it is not possible to draw concrete conclusions about underlying algorithms based on competition scores and solo scores because because the difference in scores may be because of other factors like the input representation used, the number of evaluations, and other evolution parameters.

The NEAT based approach was implemented with a goal to evolve advanced skills and strategic behavior. In addition to evolving advanced skills and strategic behavior, NEAT based controllers also achieved credible solo and competition scores comparable to some of the winning controllers from the car-racing competitions. The comparative study shows that NEAT and Neuroevolution in general is a promising approach for developing strategic controllers. More investigation is necessary to ascertain the relative abilities of CoSyNE and NEAT to evolve strategy.

## 7.3 Future Work

The work presented in this thesis provides several avenues for future research. Some of them are outlined below.

First, neuroevolution has difficulty overcoming the orbiting phenomenon. This phenomenon happens when the controller accelerates while turning towards a waypoint. The turning radius becomes large and the grip becomes low when the car accelerates, and consequently the controller misses the waypoint. If the car doesn't stop or slow down, there is a chance of this event recurring and the controller orbiting the waypoint. Once the controller starts orbiting, no more waypoints can be captured, unless the race involves an opponent who is not orbiting. Neuroevolution wasn't able to overcome this problem because orbiting did not happen often enough for evolution to consider as a serious setback. Many of the entries to the CEC 2007 competition[29], worked around this by having a hand-coded mechanism to detect orbiting and taking appropriate actions when necessary. Such measures are sometimes necessary to solve problems in the real world. An analogous approach using Neuroevolution would be to evolve a separate module that detects orbiting.

Second, though the modular approach was successful in evolving strategic behavior, it does involve a human element in the design of the modules. The next logical step would be to try to evolve the modules automatically. A mechanism was proposed by Reisinger and Miikkulainen[21] for coevolving reusable modules for high dimensional search spaces. This mechanism can be applied to the simplerace domain in order to evolve the modules required automatically. Evolution may be able to discover smaller and finer decompositions than the design used here.

Third, coevolutionary approaches were able to discover some interesting skills, but they were not able to discover any high level strategies for the car-racing domain. This is surprising because competitive coevolution is expected to result in evolution of increasingly sophisticated solutions. In [9], Kohl and Miikkulainen stated that high-level strategy problems that require the integration of multiple sub-behaviors are difficult for neuroevolution to solve. The authors consider the simplerace domain to be a fractured domain. Fractured domains are domains where the correct action varies discontinuously as the agent moves from state to state.They propose a modification to the NEAT algorithm that uses Radial Basis Function (RBF) to overcome this problem. It may prove worthwhile to implement the four approaches, particularly the coevolutionary approach using this modified version of NEAT.

Fourth, learning to bump the opponent is an useful skill, and bumping the other opponent in order to get to a waypoint first is an example of a strategy. Initial experiments show that it is possible to evolve controllers which can bump the opponent. These controllers however, were not able to leverage the skill to capture waypoints. The modular approach outlined before could be adapted to have bumping the opponent as one of the choices in addition to target waypoint selection. Learning when to bump the opponent will provide a strategic advantage to the player.

Fifth, in the simplerace domain, controllers were not penalized for having slow response times. In the real world, response time is a critical issue. One factor which affects the response time is the size of the evolved NEAT network. Though NEAT is set up to grow networks incrementally, compact networks can be rewarded to explicitly encourage evolution of smaller networks. Another way, would be to penalize large networks, based on the number of nodes and connections. Evolving smaller networks will reduce the response time of NEAT based controllers.

Sixth, evaluation in Torcs domain can validate the results obtained in simplerace domain. Torcs is an open-source car racing simulation. It features different cars, around 20 tracks, and 50 opponents to race against. It is more advanced in terms of dynamics, collision handling and graphics. It also has gear shifting. The disadvantage is that TORCS is not designed for learning algorithms. Currently, work is being done to make TORCS more suitable for learning using a client-server setup. Once completed, torcs should prove to be the logical successor to simplerace domain for evaluating the above approaches. Multiple opponents and tracks will provide more opportunities for strategic behavior.

# Chapter 8

# Conclusion

Developing controllers capable of strategic behavior for physical simulation domains is a challenging problem. This thesis had two specific goals : 1) to demonstrate that Neuroevolution can be used to evolve controllers for physical simulation domains and 2)to show that Neuroevolution and NEAT in particular, can evolve advanced skills and high-level strategy for such domains.

Four different approaches to evolution were studied and their strengths compared. Controllers developed using direct evolution were able to learn the basic skills of driving and some advanced skills. In the simplerace domain, direct evolution was implemented as solo racing. The controllers developed using this approach were able to race solo, but were not successful while racing with opponents.

An Incremental approach with refreshing was shown to evolve the necessary skills needed to compete with an opponent. Refreshing was needed to ensure that the controller retained the skills learned in previous stages. The disadvantage of this approach was that the population started to stagnate after the individuals learned to drive against the selected opponent. A pedagogy of increasingly sophisticated opponents was needed to ensure progress of evolution.

Competitive coevolution was implemented to create such a pedagogy of opponents. Since the simplerace domain was a symmetrical domain, the same population was used to evolve the hosts and the parasites (opponents). Controllers of increasing sophistication were produced by competitive coevolution. Consequently, coevolutionary approaches were able to discover skills that are hard to code by hand. However, competitive coevolution was not able to discover strategic high-level behavior in the simplerace domain.

Evolution of high-level strategy for physical simulation domains is hard for NEAT. For the simplerace domain, the problem was decomposed into two relatively independent tasks. With such a decomposition in place, modular neuroevolution was successful in evolving controllers capable of skill and strategy. The decomposition of the car racing problem into two independent tasks i.e. way point selection and navigation required domain knowledge. It remains to be seen if such a decomposition can be achieved automatically by evolution.

In summary, Neuroevolution is certainly capable of evolving advanced driving skills physical simulation domains like car racing. However, it is hard to evolve high-level strategic behavior in such domains. This thesis demonstrates that given a appropriate task decomposition, modular neuroevolutionary approaches can evolve high-level strategy. The results obtained in the car-racing domain suggest that the modular approach can be applied to evolve strategic behavior in other physical simulation domains and tasks.

# Appendix

# Appendix 1

# Simplerace Domain and Parameters

The appendix has two sections. The first section describes the NEAT parameters used for the various experiments. The second section describes the dynamics of the simplerace domain.

## 1.1 NEAT Parameters

| Parameter | Value in Simplerace domain |
|---|---|
| Population Size | 200 |
| C1 | 1 |
| C2 | 1 |
| C3 | 3 |
| Ct | 15 |
| Mutate Only Probability | 0.25 |
| Mate By Choosing Probability | 0.6 |
| Mate By Averaging Probability | 0.4 |
| Mate Only Probability | 0.2 |
| Recurrent Connection Probability | 0.2 |
| Weight Mutation Power | 2.5 |

Table 1.1: NEAT Parameters

## 1.2 Simplerace Domain

This section provides detailed information about the simplerace domain. In 1.2.1, the sensor model of the simplerace domain is described. 1.2.2 lists the equations that govern the simplerace domain.

### 1.2.1 Sensor Model

There is a slight difference in the convention used to refer to the sensors in the simplerace java package and this thesis. The currently active waypoint is called as ‘Next Waypoint’ in the simplerace package, but is referred to as the‘current waypoint’ in the thesis. Similarly, the second waypoint is called as ‘Next Next Waypoint’ in the simplerace domain, but is referred to as the ‘next’ or ‘second’ waypoint in this thesis.

| S.No | Sensor Name | Sensor Type | Use |
|---|---|---|---|
| 1 | Speed | First Person | Speed of first player |
| 2 | Angle To Next Waypoint | First Person | Angle between first player and current waypoint |
| 3 | Distance To Next Waypoint | First Person | Distance between first player and current waypoint |
| 4 | Angle To Next Next Waypoint | First Person | Angle between first player and second waypoint |
| 5 | Distance To Next Next Waypoint | First Person | Distance between first player and the second waypoint |
| 6 | Angle To Other Vehicle | First Person | Angle between first player and second player |
| 7 | Distance To Other Vehicle | First Person | Distance between first player and second player |
| 8 | Other Vehicle Is Present | First Person | This input is true if other vehicle is present |
| 9 | Position | Third Person | x,y co-ordinates of first player |
| 10 | Velocity | Third Person | velocity of first player in vector form |
| 11 | Orientation | Third Person | orientation of the first player |
| 12 | Other Vehicle Position | Third Person | x,y co-ordinates of second player |
| 13 | Other Vehicle Velocity | Third Person | velocity of second player in vector form |
| 14 | Other Vehicle Orientation | Third Person | orientation of the second player |
| 15 | Next Waypoint Position | Third Person | x,y co-ordinates of second waypoint |
| 16 | Next Next Waypoint Position | Third Person | x,y co-ordinates of second way point |

Table 1.2: Sensor Model of the Simplerace Package

### 1.2.2 Dynamics

The state of the car in the simplerace domain is represented by its position(s), velocity (v), orientation ($\theta$) and angular velocity ($\dot{\theta}$). The simulation is updated 20 times per second in simulated time and at each timestep, the state of the car is updated. The following four equations govern the update of state in simplerace domain[29].

$$s_{t+1} = s_t + v_t \tag{1.1}$$

$$v_{t+1} = v_t(1 - c_{drag}) + f_{driving} + f_{grip} \tag{1.2}$$

$$\theta_{t+1} = \theta_t + \dot{\theta} \tag{1.3}$$

$$\dot{\theta_{t+1}} = f_{traction}(f_{steering}() - \dot{\theta}_t) \tag{1.4}$$

where,

$t$ denotes the time

$c_{drag}$ is a scalar constant set to 0.1

$f_{driving}$ equals 4 for forward mode, 2 for backward mode and 0 for neutral.

$f_{traction}$ limits the change in angular velocity to between -0.2 and 0.2

$f_{steering}()$ is defined as $mag(v)$ if steering command is left and is defined as $-mag(v)$ if steering command is right and 0 if it is center.

$f_{grip}$ represents the effort from tyres to stop skidding.

# Bibliography

[1] Bobby Bryant and Risto Miikkulainen. Acquiring visibly intelligent behavior with example-guided neuroevolution. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, 2007.

[2] David Carmel and Shaul Markovitch. Incorporating opponent models into adversary search. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 120–125. AAAI, 1996.

[3] Thomas D'Silva, Roy Janik, Michael Chrien, Kenneth Stanley, and Risto Miikkulainen. Retaining learned behavior during real-time neuroevolution. *Artificial Intelligence and Interactive Digital Entertainment*, 2005.

[4] Sevan Ficici. *Solution Concepts in Coevolutionary Algorithms.* PhD thesis, Brandeis University, 2004.

[5] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[6] Faustino Gomez and Risto Miikkulainen. Active guidance for a finless rocket using neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 2084–2095, 2003.

[7] Faustino Gomez and Risto Miikkulainen. Efficient non-linear control through neuroevolution. In *Proceedings of the European Conference on Machine Learning*, volume 5, pages 317–342, 2006.

[8] J.D.Lohn, D.S.Linden, G.S.Hornby, W.F.Kraus, A.Rodriguez, and S.Seufert. Evolutionary design of an x-band antenna for nasa's space technology 5 mission. In *Proc. 2004 IEEE Antenna and Propagation Society International Symposium and USNC/URSI National Radio Science Meeting*, volume 3, pages 2313–2316, 2004.

[9] Nate Kohl and Risto Miikkulainen. Evolving neural networks for fractured domains. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1405–1412, 2008.

[10] Nate Kohl, Kenneth O. Stanley, Risto Miikkulainen, Michael Samples, and Rini Sherony. Evolving a real-world vehicle warning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2006.

[11] Alex Lubberts and Risto Miikkulainen. Co-evolving a go-playing neural network. In *Coevolution: Turning Adaptive Algorithms Upon Themselves, Birds-of-a-Feather Workshop, Genetic and Evolutionary Computation Conference (GECCO-2001)*, 2001.

[12] Simon Lucas. Computational intelligence and games:challenges and opportunities. *International Journal of Automation and Computing*, 5:45–57, 2008.

[13] Risto Miikkulainen, Bobby D. Bryant, Ryan Cornelius, Igor V. Karpov, Kenneth O. Stanley, and Chern Han Yong. Computational intelligence in games. In *In Gary Y. Yen and David B. Fogel (editors), Computational Intelligence: Principles and Practice, IEEE Computational Intelligence Society.*, pages 155–191, 2007.

[14] German Monroy, Kenneth Stanley, and Risto Miikkulainen. Coevolution of neural networks using a layered pareto archive. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 329–336, 2006.

[15] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999.

[16] Jacob Murre. *Learning and Categorization in Modular Neural Networks.* L.Erlbaum Associates, 1992.

[17] Sethuraman Muthuraman, Grant Maxwell, and Christopher MacLeod. The evolution of modular artificial neural networks for legged robot control. In *Artificial Neural Networks and Neural Information Processing ICANN/ICONIP 2003*, page 180. Springer, 2003.

[18] Dana S. Nau. Ai game playing techniques: Are they useful for anything other than games? a synopsis of the panel discussion at iaai-98. pages 117–118, 1999.

[19] Itsuki Noda and Peter Stone. The robocup soccer server and cmunited: Implemented infrastructure for mas research. In *Revised Papers from the International Workshop on Infrastructure for Multi-Agent Systems*, pages 94–101, London, UK, 2001. Springer-Verlag.

[20] Joseph Reisinger, Erkin Bahceci, Igor Karpov, and Risto Miikkulainen. Coevolving strategies for general game playing. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2007.

[21] Joseph Reisinger, Kenneth O. Stanley, and Risto Miikkulainen. Evolving reusable neural modules. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2004.

[22] C. D. Rosin and R. K. Belew. New methods for competitive evolution. *Evolutionary Computation*, 5, 1997.

[23] Christopher D. Rosin and Richard K. Belew. Methods for competitive co-evolution: Finding opponents worth beating.

[24] Stuart Russel and Peter Norvig. *Artificial Intelligence a Modern Approach*. Pearson Education, 2003.

[25] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, 2005.

[26] Kenneth O. Stanley and Risto Miikkulainen. The dominance tournament method of monitoring progress in coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002.

[27] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[28] Kenneth O. Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.

[29] Julian Togelius, Simon Lucas, Ho Duc Than, Jonathan Garibaldi, Tomoharu Nakashima, Tan Chin Hiong, Itamar Elhanany, Shay Berant, Philip Hingston, Bob MacCallum, Thomas Haferlach, Aravind Gowrisankar, and Pete Burrow. The 2007 ieee cec simulated car racing competition. *Genetic Programming and Evolvable Machines*, 9:295–329, 2008.

[30] Khare V.R, Xin Yao, Sendhoff B., Yaochu Jin, and Wersing H. Co-evolutionary modular neural networks for automatic problem decomposition. In *The 2005 Congress on Evolutionary Computation*, volume 3, pages 2691–2698, 2005.

[31] Shimon Whiteson, Nate Kohl, Risto Miikkulainen, and Peter Stone. Evolving keepaway soccer players through task decomposition. *Machine Learning*, 59:5–30, 2005.

# Vita

Aravind Gowriankar was born in Chennai, Tamilnadu on 31 January 1984, the son of Gowrisankar and Jamuna. He received his Bachelors in Engineering degree from Anna University,Chennai in 2005. After graduation, he worked as Programmer Analyst at Cognizant Technology Solutions Ltd in Chennai . In August 2006, he started his graduate studies in Computer Science at the University of Texas at Austin. Aravind works with Professor Risto Miikkulainen at the Neural Network Research Group in UT Austin. His research interests are in Machine Learning and NeuroEvolution. In his free time, Aravind dabbles with digital photography. He is an active member of the Technology Entrepreneurship Society at UT and serves as the Vice-President of Communications.

Permanent address: 924 E. Dean Keeton St, Apt 124
Austin, Texas 78705

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.