Novelty Search in Competitive Coevolution using Normalized Compression Distance

by

David Andrew Kirkpatrick

Bachelor of Arts
Psychology, Economics
Saint Mary's College of Maryland
1999

Master of Science
Computer Information Systems
Florida Institute of Technology
2006

A thesis submitted to the
College of Engineering at
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Melbourne, Florida
May, 2012

We the undersigned committee hereby recommend
that the attached document be accepted as fulfilling in
part the requirements for the degree of
Master of Science in Computer Science.

"Novelty Search in Competitive Coevolution
using Normalized Compression Distance,"
a thesis by David Andrew Kirkpatrick.

____
D. B. Weddle, Ph.D.
Associate Professor, Computer Science
Thesis Advisor

____
D. W. Mutschler, Ph.D.
Associate Professor, Computer Science

____
V. C. Gordon, Ph.D.
Associate Professor, Aerospace Engineering

____
W. D. Shoaff, Ph.D.
Department Head, Computer Science Department

Abstract

Title:
Novelty Search
in Competitive Coevolution
using
Normalized Compression Distance

Author
David Andrew Kirkpatrick

Principle Advisor
D. B. Weddle, Ph.D.

Competitive Coevolution using Neuroevolution of Augmenting Topologies (NEAT) is a technique for evolving neural networks using genetic algorithms in problem domains where two teams are attempting to evolve increasingly sophisticated strategies to defeat each other. The genome scoring required to drive the genetic algorithm and evolutionary process has traditionally been accomplished using objective fitness measurements of the neural networks, despite research showing often substantial benefits to using a behavioral novelty metric as a method of ranking genomes. The lack of a measure of behavioral novelty in competitive coevolution is largely due to the complex nature of competitive problems, which makes the development of algorithms to measure the novelty of behaviors exceptionally difficult. To address this issue, this paper discusses the application of Normalized Compression Distance (NCD) as a general method to developing measures of novel behavior in competitive problems, and compares the performance of Novelty Search using NCD to traditional objective fitness approaches.

Table of Contents

List of Figures

List of Tables

Acknowledgement

I would like to acknowledge and thank the advice and guidance of Dr. Danny Weddle, my thesis advisor. I also thank the members of my graduate committee for their guidance and their time spent supporting my research efforts.

I would like to acknowledge and thank Dr. Kenneth Stanley and Dr. Rudi Cilibrasi for their advice and guidance on the topics of neuroevolution, competitive coevolution, and normalized compression distance.

I acknowledge my supervisor Pat Gatewood, who has ever been supportive of my graduate studies, and who has allowed me to utilize computer resources at work to run my experiments. I also thank him for teaching me to program when I was starting my career in human factors psychology, and sending me on a very interesting and unexpected career path.

I would also like to thank my wife, Dr. Janet Nye, for all her support through the years, and her seemingly endless supply of patience.

Dedication

I would like to dedicate this thesis to my loving wife Janet and to our newborn daughter Hope. Thank you for all of your patience and support, and listening to my crazy ideas.

Chapter 1
Introduction and Background

Competitive Coevolution using Neuroevolution of Augmenting Topologies (NEAT) is a technique for evolving neural networks using genetic algorithms in problem domains that require that the fitness of a neural network be evaluated through competition with other neural networks in the population. By applying NEAT genetic algorithms to a competitive problem domain, the researchers' goal is to start an "arms race" between two opposing sides, in which ever increasingly effective strategies are evolved as each team develops new tactics to win, and those tactics are then bested by the opposing team. In order for the NEAT genetic algorithm to function, a ranking or score of a neural network must be provided to the genetic algorithm prior to starting the next step in evolution. Traditionally this ranking is based on objective fitness of a neural network in the task being undertaken. Essentially, the better the neural network is at performing the task, the higher its score.

Recent studies have shown excellent results by ranking neural networks based on the novelty of their behavior as opposed to their objective fitness, and by using this "novelty search" approach, highly deceptive tasks that would be difficult or impossible to solve using an objective fitness approach are now possible (Lehman & Stanley, Exploiting Open-Endedness to Solve Problems Through the Search for Novelty, 2008). Unfortunately, novelty search has not been applied to competitive coevolutionary problems previously due to the complexity of developing a metric for novel behavior in the highly dynamic tasks that involve interaction with other neural networks. Attempts to break down the complex interaction between two neural networks would involve significant expert knowledge of the problem domain and would prove time consuming.

In order to expand the viability of competitive coevolution in a broader range of problem domains, a method for applying novelty search to competitive coevolutionary problems in a general way that does not require expert knowledge of the problem domain would prove useful. Previous studies have shown promise applying Normalized Compression Distance (NCD) to sustaining diversity through novelty search in neuroevolution problems (Gomez, 2009). Given previous successes utilizing NCD as the foundation for a general novelty metric, the author of this thesis hypothesizes that NCD will also work as foundation for a novelty metric in competitive coevolutionary problem domains, despite the added complexity of interactions between neural networks.

This thesis describes the implementation of a competitive coevolution experiment that supports both objective fitness ranking, as well as various implementations of novelty search using NCD as a general approach to measuring behavioral novelty that does not require expert knowledge of the problem domain. The robot duel task in the experiment is modeled closely after that utilized in the paper on competitive coevolution using NEAT (Stanley & Miikkulainen, Competitive Coevolution through Evolutionary Complexification, 2004). Results of the various conditions tested are presented as well as a discussion of the feasibility of applying NCD to measure behavioral novelty in competitive coevolutionary problems.

Neuroevolution of Augmenting Topologies

NEAT is a method of neuroevolution that applies genetic algorithms to the development of neural networks in such a way that not only are the activation weights of the individual neurons in the network evolved, but also the network architecture, or topology. In any neuroevolution system, the way in which the evolutionary genetics are encoded is of particular importance. The evolutionary genetics are represented by “genomes” in NEAT, which are linear representations of the network connectivity. The genes in the genome provide all of the information needed to create the neural network. Two additional critical features of NEAT genomes is that they provide the ability to cross-over two parent genomes to create an offspring genome that inherits characteristics of the two parent genomes, and they provide a means of calculating genome compatibility. The measure of genome compatibility allows for speciation, or the grouping of genomes into similar species. Such a mechanism allows for varied approaches for handling diversity, and protecting new species from dying off too early before they have had time to develop. The individual genes that make up the genome represent either neurons or links between neurons in the overall network described by the genome (Stanley & Miikkulainen, Evolving Neural Networks through Augmenting Topologies, 2002).

One of the most important aspects of NEAT that differentiates it from other neuroevolution approaches is the inclusion of innovation tracking, which directly influences almost all major functionalities in the evolutionary and speciation algorithms (Stanley & Miikkulainen, Evolving Neural Networks through Augmenting Topologies, 2002). When a new link or neuron gene is added to the structure represented by a genome, it is checked against a database of prior innovations to see if it is a new structural innovation, or if it has occurred already. New innovations receive a new innovation number, but innovations that match an existing innovation share the same innovation number. This approach allows NEAT to keep a historical record of structural changes, and is essential in the design of a valid crossover operation between genomes. The crossover operation involves aligning similar genomes based on their innovation numbers (Figure 1). Genes that match between the two genomes are inherited by offspring randomly (the genes represent the same structure, but the activation weights could still differ). The non-matching (disjoint and excess) genes are inherited from the “fitter” or better performing of the two parents. This mechanism ensures that offspring are valid, and retain capabilities of the parent genomes (Buckland, 2002).

It is important to keep in mind that NEAT is capable of evolving complex neural networks, including recurrent connections that allow for the development of memory. Many tasks to which a neural network may be applied to are only capable with, or made significantly easier with the ability to form memory. As NEAT evolves neural networks (which is performed between generations), it slowly increases the topology. This slow growth increases the chances of finding a more optimal topology than an explicitly designed topology, such as would be found in traditional feed-forward hidden layer neural networks. As the topology increases slowly, NEAT has the capability to optimize the network at each successive step in its development, which minimizes the dimensions of search, and speeds up the overall search process (Stanley & Miikkulainen, Evolving Neural Networks through Augmenting Topologies, 2002)

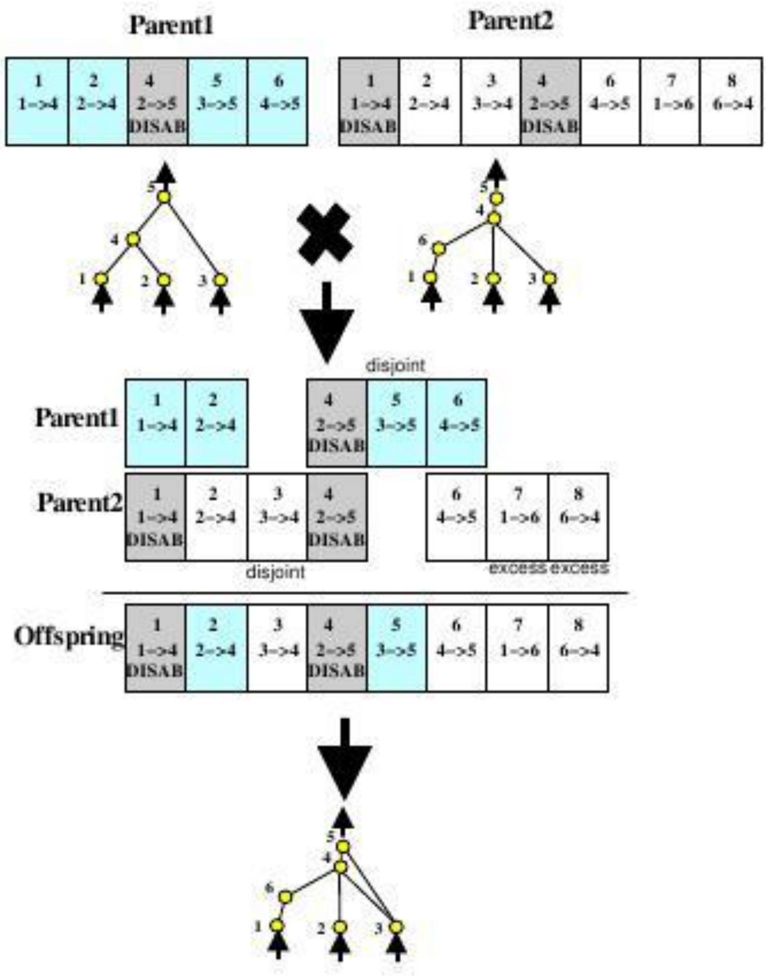

Figure 1: Matching of genomes from two parents prior to the crossover operation. The top number represents the innovation number for the gene. Both parents share genes one, two, four, and six. From parent one, gene five is disjoint. From parent two, gene three is disjoint, and genes seven and eight are excess (Stanley & Miikkulainen, Evolving Neural Networks through Augmenting Topologies, 2002).

When compared to a more traditional backpropagation approach to neural network learning that requires supervised training, NEAT is capable of unsupervised training, opening even more problem domains that can be approached using this method. In the case of training a neural network with backpropagation, inputs are given to the neural network, and the desired outputs based on those inputs. The neural network makes modifications to its weights based on this desired output in order to “learn”. Of course this assumes that the researcher knows what output is desired given the inputs fed into the neural network. This may not be the case in all problem domains. With a neuroevolution system such as NEAT, the desired neural network outputs do not need to be provided, only a higher level rating function that drives the genetic algorithm. A greater range of complex problems can be solved using this unsupervised training approach.

Competitive Coevolution

In the objective fitness approach to using NEAT, the closer a neural network gets to the solution in a task, the higher its fitness ranking. Ranking neural networks based on their progress towards a predetermined goal is a logical approach in an environment in which the neural network is not competing with other neural networks. However, if the goal of the system is to evolve neural networks that can successfully compete against other neural networks a different approach is required, as objective fitness measures will not be absolute compared to a set goal, but relative to the performance of the rest of the population.

Competitive coevolution is such an approach, and works by ranking the fitness of individual neural networks through competition with other individuals in the population instead of through an absolute fitness measure. The competition based scoring of neural networks in competitive coevolution is not an absolute fitness measure since the fitness ranking represents the relative strength of a particular neural network when compared against its competitors. In the competitions the neural network being evaluated is traditionally called the “host”, while the carefully selected group of opponents is referred to as the “parasites” (Rosin & Belew, 1997). The selection of “parasites” should be done in such that they are representative of higher level capabilities of the other neural networks in the population, or in the case of this experiment, from the other team. In order for interesting strategies to be evolved over time an “arms race” must be established within the simulation for a significant large number of generations (Stanley & Miikkulainen, Competitive Coevolution through Evolutionary Complexification, 2004). If an arms race can be sustained over many generations, the two opposing sides will evolve increasingly effective strategies to defeat their opponents.

Competitive evolution is a valuable approach when interactive behaviors need to be evolved for which it would be difficult to define an absolute fitness function for, or when development of opponents to compete against would be a non-trivial or time-consuming task. Competitive coevolution helps to solve this problem by splitting the neural network population in half, and alternating which side is being evaluated (hosts), and which side is providing the opponents (parasites), allowing the quality of the opponents to increase over time, and freeing the researcher from needing to code the opponents by hand. Since the neuroevolution process requires large populations to be evaluated and evolved over numerous generations, opponents and their behavior need to be modeled so they can be run in accelerated simulations. The evolutionary process tends to find the simplest solution that wins, meaning that without careful management the two opposing sides would simply switch back and forth between the same few uninteresting strategies (Rosin & Belew, 1997). By carefully selecting appropriate parasites, the problem of opposing sides switching back and forth between few uninteresting strategies can be avoided.

What is desired is not that neural networks should evolve to defeat the best strategies of the current population of opponents, but evolve to be capable of defeating those current strategies as well as the best strategies from previous generations. In order to accomplish this task, a "hall of fame" of best neural networks is often drawn from for parasite opponents. By utilizing NEAT as the neuroevolution system, the arms race process is further encouraged as the topology of neural networks grow slowly, allowing new dimensions in the search space for strategies even if a temporary global optimum is reached (Stanley & Miikkulainen, Competitive Coevolution through Evolutionary Complexification, 2004).

Novelty Search

Objective based NEAT has a tendency to occasionally become stuck in local optima, and to fail or struggle in tasks that exhibit deception. This is due to the fact that an objective based fitness function does not necessarily reward the stepping stones in the search space that would ultimately lead to the objective. In order to reach the solution in deceptive problems, the neural networks must move further away from the goal, in order to ultimately reach it. Using a simple fitness function that rewards based on distance from the goal would likely fail or perform poorly in this case. A simple example is the hard maze navigation map (Figure 2) where one must get further away from the objective goal as a stepping stone in order to ultimately reach the end goal.

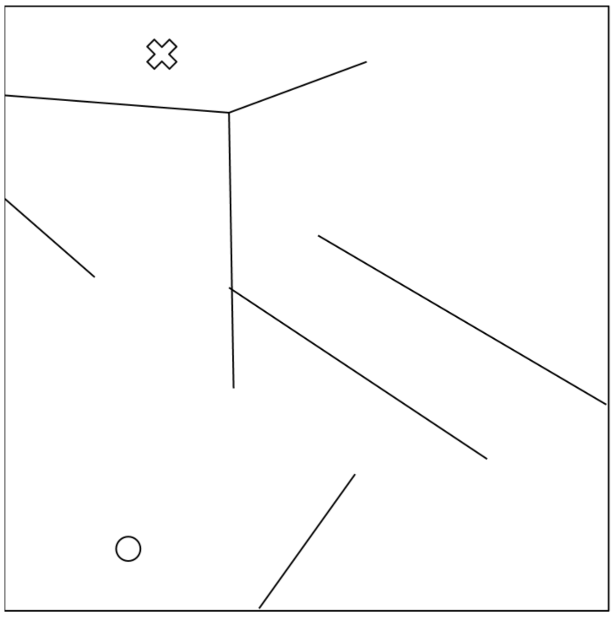

Figure 2: Hard Maze example, with the X being the goal, and the circle the starting point of the robot. With an objective fitness function that rewards robots based on how close they end the task to the goal, robots will tend to get stuck in a cul-de-sac (Lehman & Stanley, Exploiting Open-Endedness to Solve Problems Through the Search for Novelty, 2008).

As NEAT is applied to ever increasingly complex problems, deceptive fitness landscapes become more likely, as the more ambitious the goal, the more likely it is that the search can be deceived and trapped in local optima. In a task with an ambitious goal, a carefully scripted series of stepping stones towards the ultimate objective must be created, requiring intimate domain knowledge, careful oversight, and the identification of these stepping stones *a priori*. The application of novelty search serves as a proxy to these stepping stones, greatly simplifying the process of developing systems to tackle complex problems (Lehman & Stanley, Exploiting Open-Endedness to Solve Problems Through the Search for Novelty, 2008).

Fortunately, evolutionary algorithms such as NEAT are relatively easy to modify in order to implement novelty search. Essentially the only modification needing to be made once a novelty metric is defined is to replace the fitness measure with the novelty measure. There are many means by which to measure differences between neural networks, and the genomes that encode them. The underlying NEAT genetic algorithm uses genome compatibility to measure the difference between the genomes themselves for the purposes of speciation. Objective fitness based NEAT uses measures of the fitness of a neural network's performance. Novelty search is yet another means, which measures the difference in the actual behavior of a neural network when compared to other neural networks in the population (Lehman & Stanley, Exploiting Open-Endedness to Solve Problems Through the Search for Novelty, 2008). The more difficult a task, the more likely the use of novelty search will provide greater ability to solve the problem at hand. Unfortunately, the more difficult the task, the more difficult it is to define a novelty metric.

Normalized Compression Distance

Given the difficultly in defining a novelty metric in complex tasks where intimate domain knowledge is required, a general method for measuring differences in behavioral novelty would prove useful. NCD is such a method as it does not use subject-specific features or background knowledge of the data it is comparing (Cilibrasi & Vitanyi, 2005), and has been successfully used to create novelty metrics in the past (Gomez, 2009). NCD has been used to correctly classify disparate file types ranging from compiled java code to Jimi Hendrix songs in midi, without any form of domain knowledge (Gomez, 2009). It is important to keep in mind that the function of NCD operates by compressing data, so it is critical that data being compared not be compressed prior to measurement. NCD is defined by the equation

$$NCD(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}$$

Where *NCD*(*x, y*) is the normalized information distance between data *x* and data *y*, *C* is a compressor to be used, *C*(*xy*) denotes the compressed size of the concatenation of *x* and *y*, *C*(*x*) denotes the compressed size of *x*, and *C*(*y*) denotes the compressed size of *y* (Cilibrasi & Vitanyi, 2005). The normalized information distance measure will be roughly between 0.0 and 1.0. The smaller the value, the more similar the two objects are.

The use of a compression algorithm as a measure of informational distance depends on the ability to compress one object given the data contained in the other object. The more similar the two objects, the more compression we can achieve in the concatenation (Cilibrasi & Vitanyi, 2005). While not a perfect measure of difference between the two data objects, even a rough measure of difference between two objects is strong enough to warrant its use given the flexibility and simplicity of applying a general approach to measuring information distance. While NCD has been shown to be a successful approach to developing novelty metrics in the past (Gomez, 2009), it has not been applied to the more complex competitive coevolutionary problem domain until now.

Chapter 2
Software Design

The software that makes up this experiment is written in C++ and was developed and run on Linux computer systems. As the source code to the competitive coevolution experiment by Stanley and Miikkulainen was not available, the experiment described in the following sections is modeled on, but not identical to their competitive coevolution experiment. The most critical differences between the implementation described in this thesis and the original is the use of a different implementation of NEAT, different parameters used in the genetic algorithm, and the inclusion of numerous novelty search experimental conditions. Reimplementing the original experiment proved a time consuming undertaking, and in the hopes that future efforts can be saved for researchers interesting in working with competitive coevolution, all of the source code to the various programs that make up this thesis research effort will be released under the BSD software license. With the reusable class based design of this software, it is hoped that all or parts of this software will prove useful in the future to others.

Programs Developed

There are three software programs that were developed to support this thesis research effort: the Coev GUI, Genome Tester, and Competitive Coevolution Experiment software. Running neuroevolution experiments can be a computationally expensive undertaking, especially with the use of NCD as will be illustrated later in the paper. Numerous multi-processor Linux computer systems were utilized to run experiments in parallel. These systems were at a remote location and only accessible through a Virtual Private Network (VPN). The Coev GUI application was intended to be a client side user interface that could connect to a running experiment to monitor the experimental progress, display statistics in real time, and provide various visualization options. This software was quickly abandoned as the amount of data produced in real-time by as many as nine parallel evaluations running per computer at accelerated rates proved to be overwhelming, and a post-experiment analysis approach was adopted. The source code for the abandoned Coev GUI is included in the software, and the network server code has been left in the experiment software in order to give future developers a starting point if they desire such functionality. All network traffic is over TCP, and the network code and user interface for the client software use the open source QT Cross Platform User Interface framework and development libraries.

The Genome Tester and Competitive Coevolution Experiment programs were actively used to carry out this research effort. The Genome Tester application is a simple OpenGL based program developed using FreeGlut libraries that is used primarily to recreate the matches and tournaments used in the experimental software, while visualizing the game board and how the robots controlled by the neural networks play the game. Genomes saved as files during experimental runs can be loaded with the Genome Tester, and the neural networks described by their genes used as robotic controllers just as they are in the experiment software. This serves to not only allow for testing repeatability of the match results seen in the experiments, but also a more careful examination of the behavior of particular neural networks. Since the Genome Tester software is separate from the experiment, it allows neural networks evolved in one experimental condition to compete against neural networks evolved in other conditions, a feature that was used to run a round-robin tournament comparison of dominant strategies from all conditions. Additionally, the Genome Tester can also be used to visualize the structure of the neural network, as well as to replay recordings of matches, a feature added solely to support development testing and debugging.

The primary effort of this project was in the development of the actual experiment software. This Competitive Coevolution software encompasses the NEAT implementation, the NCD implementations, data collection methods, the job scheduling and results processing thread, the job execution threads, and the network server thread that handles connections from the Coev GUI application (Figure 3). Given the large number of remotely located multi-processor Linux computers available to run experiments for this thesis, a command-line only, multi-threaded approach was used to develop the software. A collection of four-core, four-core hyper-threaded, and eight-core computer systems were used whose specifications can be seen in the appendix. The number of job execution threads created to run evaluations is a variable controlled in the experimental parameters file, and in general an n+1 approach was used, spawning five execution processes on four-core machines, and nine execution processes on eight-core and 4-core with hyper-threading machines.

The two most important aspects of this software are the job manager and job execution threads, which communicate through shared memory protected by a single semaphore lock. The job manager thread is responsible for scheduling the various evaluations and calculation jobs, processing the results from the evaluations, handling the interactions with the NEAT genetic algorithm, and recording data. The job execution threads retrieve a few jobs at a time to execute from a global job queue controlled by the job manger, executes them, and places the results onto a queue for the job manager to handle. While all of the genetic algorithm functions are running in the single thread of the job manager, the vast majority of computation is involved in running the evaluations and calculating NCD values, so this simple division of labor works well to leverage the hardware available in a typical multi-processor computer system.

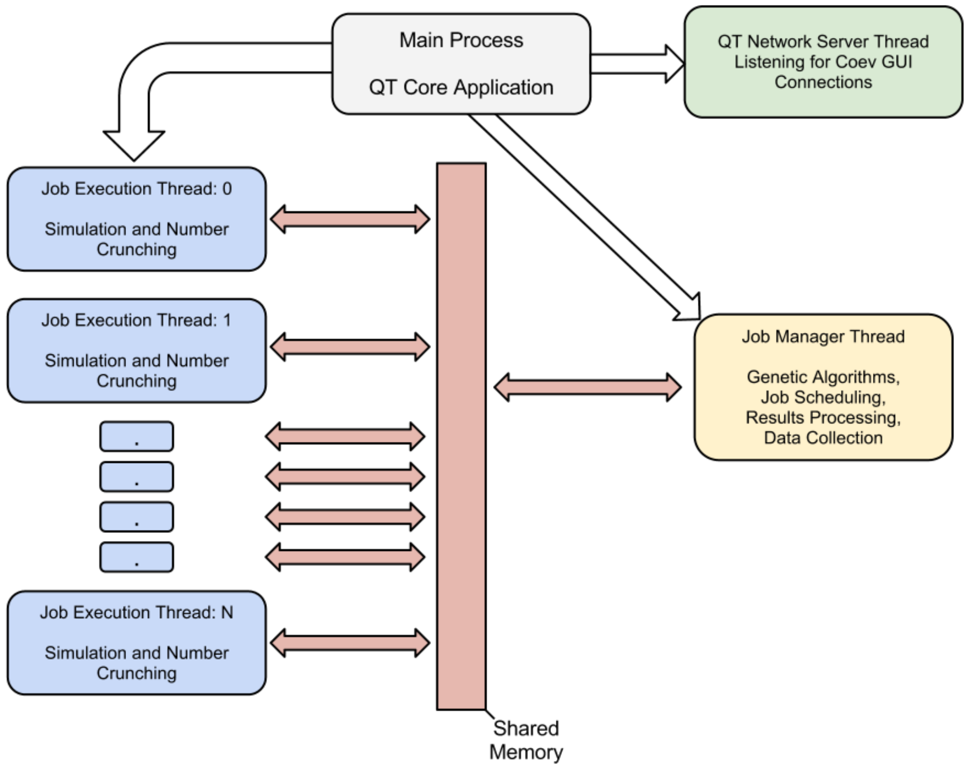
Main Process
QT Core Application
QT Network Server Thread Listening for Coev GUI Connections
Job Execution Thread: 0
Simulation and Number Crunching
Job Execution Thread: 1
Simulation and Number Crunching
.
.
.
.
Job Execution Thread: N
Simulation and Number Crunching
Job Manager Thread
Genetic Algorithms, Job Scheduling, Results Processing, Data Collection
Shared Memory

Figure 3: Process organization of Competitive Coevolution software

In order to support analysis of the results of the experiment, a number of data collection mechanisms were added to the software. Statistics about the current generation were output on the command line for easy remote monitoring, and were saved as experimental log files. While useful for quickly checking the status of an experimental run, these log files were not used for the actual analysis. The variables needed for analysis were written in comma separated values format (csv), with each line representing a single generation in the experimental run. These csv files allow for easy import into any number of spreadsheet and analysis programs for examination and further analysis. With the experiments running at accelerated rates, a means of saving interesting genomes for later investigation was needed. A method for saving a genome to, and reading from, a file was added to the underlying genome class, and particularly interesting genomes were saved for later use with the Genome Tester application. The last method of data collection that was only used for testing and debugging purposes was the saving of rendering data of matches. All analysis described in this thesis were performed with the saved genome files, and the csv data files.

The Robot Duel Domain

The problem domain used in the experiment is modeled closely after that used by Stanley and Miikkulainen in their original NEAT competitive coevolution experiment. Two robots compete on a 600x600 game board with a number of food pieces scattered around the board (Figure 4). The robots start facing away from each other, with one facing left and one facing right. As the robots move forward and turn they use energy, and the faster they move and turn, the faster they deplete their energy supply. By running over food pieces robots receive a boost to their energy, and remove the food piece from the board for the duration of the match. Robots never run out of energy in a game, but if the robots collide, the robot with the higher energy level wins. If the robots do not collide with each other after 750 time steps (30 seconds real-time) the game ends with no winner. This robot duel is a particularly interesting problem domain as it combines predator, prey, and foraging tasks, with the ability to rapidly switch between the tasks based on the current conditions in the game (Stanley & Miikkulainen, Competitive Coevolution through Evolutionary Complexification, 2004).

In standard evaluations between two robots, two games are played with nine food items placed around the game board (Figure 4). With the food layout being asymmetrical, the robots switch starting positions between the two matches. In addition to the standard evaluations, there are tournament matches between two robots that involve 158 games being played between the two robots on boards with asymmetrical food layouts. There are 79 different food configurations used in these tournament matches, with robots repeating matches after switching the side of the board they start on. For these more extensive tournament matches the original nine piece food layout is used, as well as a single extra food piece in one of twelve different food positions, and then two extra food pieces in the various combinations of the 12 extra food piece positions.

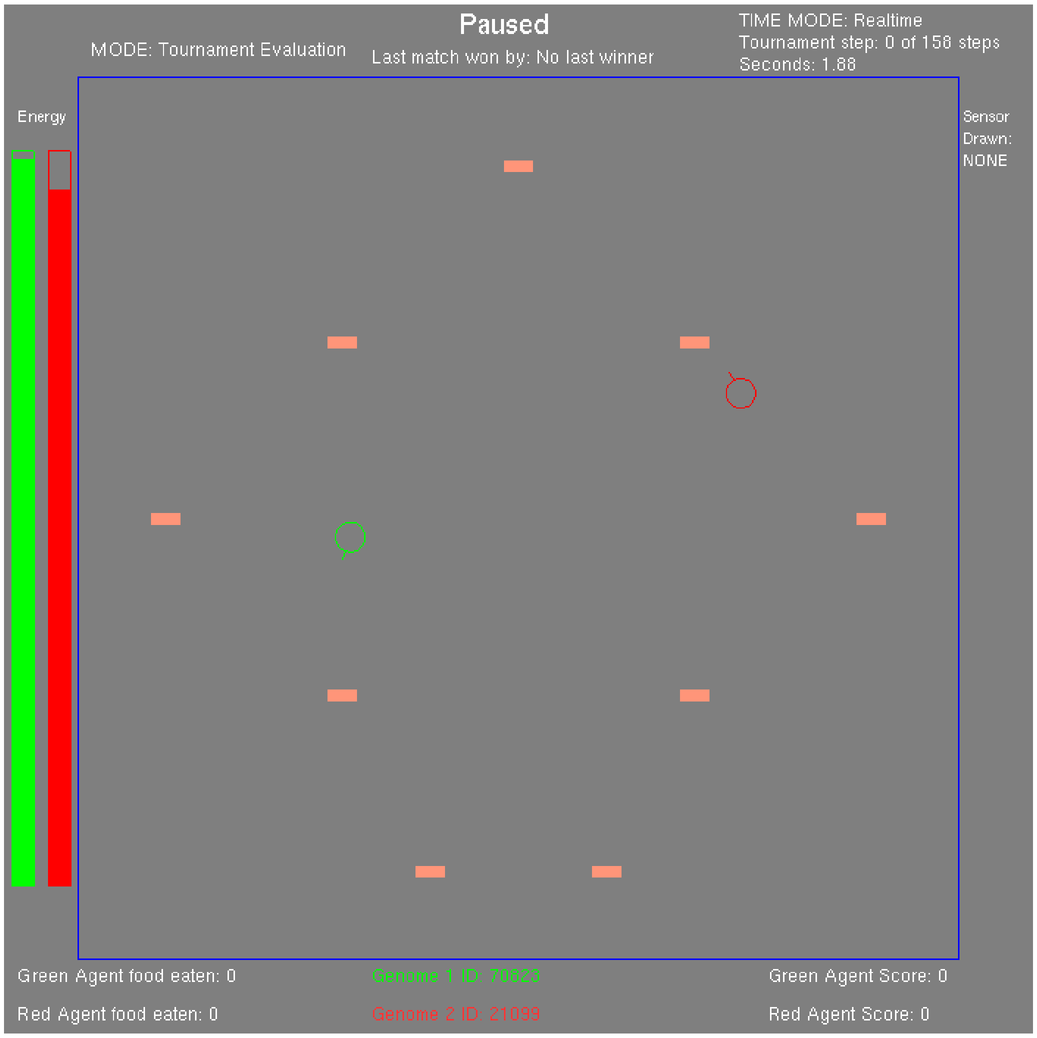
Paused
MODE: Tournament Evaluation
Last match won by: No last winner
TIME MODE: Realtime
Tournament step: 0 of 158 steps
Seconds: 1.88
Sensor Drawn: NONE
Green Agent food eaten: 0
Genome 1 ID: 70823
Green Agent Score: 0
Red Agent food eaten: 0
Genome 2 ID: 21099
Red Agent Score: 0

Figure 4: Screenshot of game board as rendered in the Genome Tester software.

The robots themselves have 13 inputs representing 12 sensors and a single bias input value, and three outputs. The first five sensors allow the robot to perceive direction and distance to food items around it. The next five sensors are similar to the direction and distance sensors for food, but represent the distance and direction of the enemy robot. The 11th sensor represents the distance to the wall directly in front of the robot, and the last sensor indicates the difference in the energy levels between the robot and its enemy. The last input represents a bias level that is left to a constant value of 1.0. There are slight differences in how the directional food and enemy sensors operate. With many food items arranged around the game board, the five food sensors allow the robot to perceive multiple food items simultaneously. These directional sensors for both food and enemy represent 30 degrees forward (15 degrees right and 15 degrees left off the nose of the robot), 75 degrees on the front sides, and the back 90 degrees on either side (Figure 5). If there are no food items within the bearing range of a sensor, the input for that particular food sensor is set at 0. If there is a food item within that sensor bearing range, the input value is set based on the distance to the food item, with values going up the closer to the food item the robot is, with an input of 1 representing a distance of 0. If multiple food items fall into the sensor range, the sensor represents the closest of the food in that sensor range.

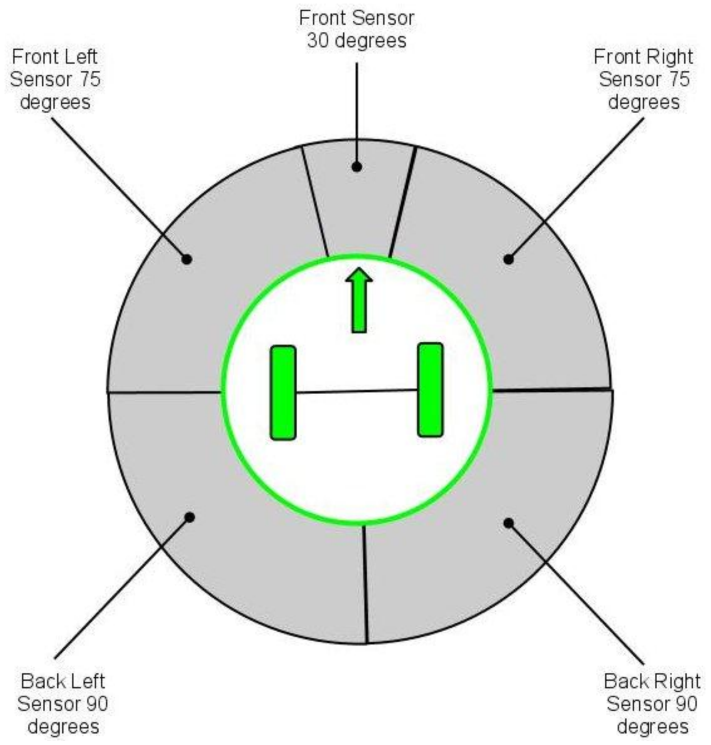
Front Sensor 30 degrees
Front Left Sensor 75 degrees
Front Right Sensor 75 degrees
Back Left Sensor 90 degrees
Back Right Sensor 90 degrees

Figure 5: There are two sets of directional sensors, one for food items, and one for the enemy. If no food or enemy fall into these relative bearing ranges, the input is 0. If the enemy or any food items do fall in these sectors, the input value represents how close the object is, with values decreasing the further away the food or enemy is.

The five enemy sensors share the same bearing ranges of 30, 75, and 90 degrees as the food sensors; however there is an additional overlap section between sensors. There is a total overlap of 10 degrees between sensors (+/- five degrees from the sensor boundaries). For example, if the enemy is at a bearing of 10 degrees, both the forward and front-right sensors will be stimulated with the distance to the enemy. If the bearing to the enemy is nine degrees, only the front sensor would be stimulated with the distance to the enemy. The enemy distance is calculated in the same manner as distance to food items (a value of one being extremely close, zero being far away or not within the sensor bearing range). The next sensor is the wall distance sensor, and represents the distance to the wall directly in front of the robot. The distance is calculated in the same manner as the food and enemy distance, with the difference that there is always a wall in front of the robot, so the input value never reaches the minimum input value of zero. The closer the robot is to the wall, the higher the input value.

The last two inputs are the energy difference sensor, and the constant bias input. The range of the energy difference sensor is from negative one to one, with a value of zero indicating the robot and its enemy has the same energy levels. The more energy the robot has than its enemy, the higher the positive value of this input, and the less energy the robot has then its enemy, the lower the negative value of this input. With all of these sensor inputs, the robot can perceive the world around it well enough to formulate strategies, however it cannot directly observe the internal state of the enemy robot, making the robot duel task a partially-observable Markov decision process (Stanley & Miikkulainen, Competitive Coevolution through Evolutionary Complexification, 2004). The three outputs from the neural networks correspond to left and right wheel settings, and the forward output power. If the right wheel output is higher than the left wheel output, the robot turns to the left, and vice versa. The higher the forward output from the neural network, the faster the robot moves forward. The faster the turning rate and speed of the robot, the more energy is consumed.

The robots and game world are implemented as C++ classes, easing their reuse in applications besides the experimental software, such as their use in the Genome Tester software, as well as their use in the numerous job execution threads. Each job execution thread instantiates its own game world type, as well as two robots to use in its simulations. Neural networks are passed to the robots to use as a robotic controller, and then the two robots are passed to the game world where the match is run. The game world handles the physics and sensor calculations, energy use by the robots, and scoring. The robot class simply pushes sensor values into the neural network, and retrieves the neural network outputs, which is used by the game world class to move the world forward one time step.

NEAT Implementation

The NEAT implementation is object oriented, with the job manager thread instantiating two instances of the genetic algorithm type Cga, one for each team in the competitive coevolution experiment. The interactions between the job manager thread and the Cga class are limited (Figure 6), with the job manager primarily providing scores to rate the genomes of a generation, triggering the next iteration of evolution when a generation is complete, asking the genetic algorithm to save interesting genomes, and retrieving the neural networks created by genomes contained in the genetic algorithm to use as robotic controllers for the next generation. While the experiment side use of the NEAT Cga class is straight forward, there is quite a bit of functionality encapsulated in the classes that make up the NEAT implementation. This NEAT implementation is based upon the Windows implementation by Mat Buckland (Buckland, 2002). The various classes of his WinNeat algorithms were modified to support their use in multi-threaded competitive coevolutionary experiments on Linux computer systems.

While the experiment software code primarily interacts with the Cga genetic algorithm class, a number of classes are needed to support the functionality of the NEAT system described here. In addition to the Cga class, the NEAT implementation consists of CGenome, CNeuralNet, CSpecies, and CInnovation classes. The essential functionality of the Cga type is to provide the neural networks of the current population, to take the scores from evaluations of those neural networks, and evolve the genomes that created the neural networks based on those scores into the next generation. The Cga class handles the underlying NEAT functionalities of speciation, innovation tracking, and performing the crossover of genomes to create offspring. The neural networks passed from the Cga class to the experiment using it are created by the CGenome class itself, as the CGenome type contains all of the information about nodes and connections needed to construct the neural network. Genomes are stored in the Cga class for the current population, a “hall-of-fame”, the leader of each species, and a custom archive which is externally controlled by the experiment software. The hall-of-fame archive by default stores the genome with the highest score from each previous generation. This default behavior can be turned off, and additions to the hall-of-fame can be manually controlled similarly to the custom archive, giving the experimental software two archives in which it can selectively store genomes for future use. The separation between the object oriented NEAT implementation and the experimental software clearly separates the reusable implementation of NEAT, and the customized code needed to implement an experiment (Figure 6).

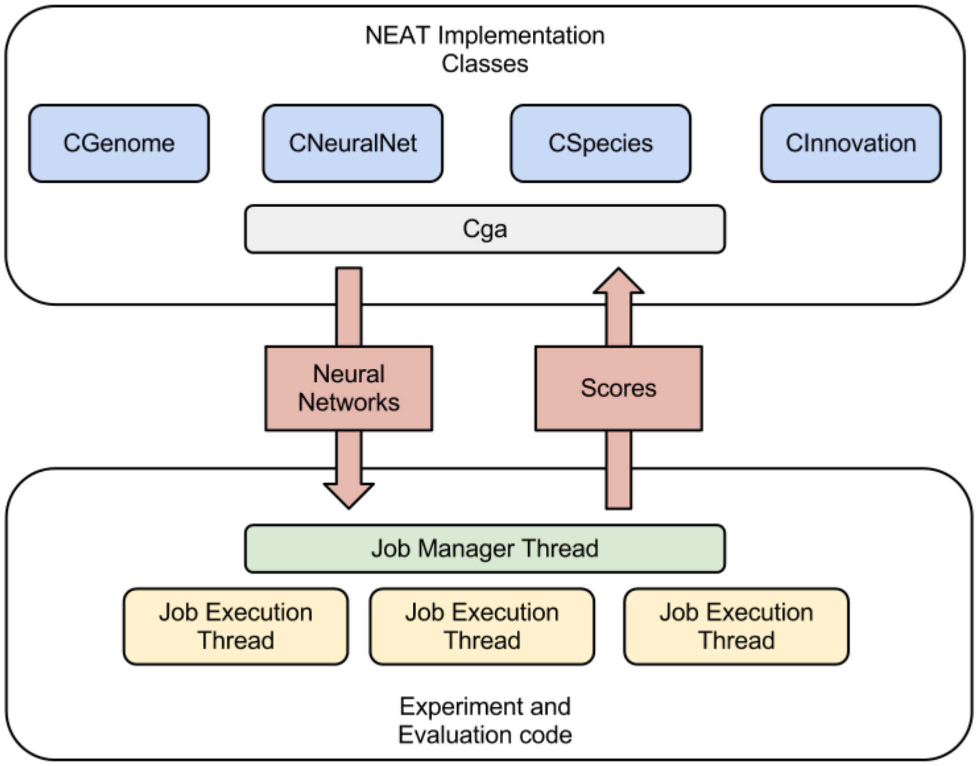

Figure 6: The primary interaction between the experiment software and the NEAT implementation. The Cga class provides neural networks for the experiment to use, and the experiment passes back the scores of those neural networks after being evaluated. This process is repeated each generation.

In the WinNeat implementation genomes are stored in vectors in the Cga class. This vector storage method is retained in the NEAT implementation used in this thesis work, but has been supplemented by an additional “vector of vectors” of genomes for instances where multiple processes will be utilizing the neural networks simultaneously. The Cga type passes neural networks back to the experiment as pointers. In order to avoid multiple processes using the same neural network simultaneously, functions were added to retrieve duplicate neural networks if needed. Each time these duplicate functions are called, a new copy of the genomes in the particular archive being accessed is created, neural networks created from the newly copied genomes, and the pointers to the duplicate neural networks are passed back to the experiment software. In practice, the experiment software calls these duplicate functions once for each execution thread being used. The neural networks contain a copy of the genome ID number from which they were created, and an additional clone ID, which increments for each set of duplicates made. At the end of a generation, all of the duplicate vectors in the Cga class are cleared, and must be recreated again from the experimental software by calling the duplicate functions once again.

The numerous parameters used to tailor the use of the NEAT system and in particular its genetic algorithms are stored in a parameters.h file. Given the seemingly endless possibilities for misconfiguring the evolutionary algorithm through these parameters, an easy method of validating the parameters was added. The use of an XOR test is commonly used in neuroevolution as a basic functional test of parameter configuration. The Cga class has a built in XOR test function, that when called takes the experiment parameters and runs a built in XOR experiment, prints the results, and exits the program. This allows the testing of parameters by adding a single line in the experiment source code. Any time parameters are changed, it is recommended to run the XOR test to ensure that the parameters being used are valid. This XOR test simply checks to see if neural networks can be consistently evolved that will output a value of one, if and only if one of its two inputs is a one. Recurrent links are disabled in XOR test mode as this makes the problem too easy to solve. Recurrent links allow a neural network to form memory, and if left enabled during XOR tests, it is possible for neural networks to simply memorize the correct order of outputs required to pass the XOR test, instead of actually solving the problem. Randomizing the inputs used in the test is one way of minimizing this problem, but the solution used in the Cga class described here is to simply disable the ability to add recurrent links during this test.

NCD Implementation

All of the approaches to implementing novelty search in the experiment rely on NCD algorithms to measure behavioral novelty, although the approaches vary by the information that is used to represent behavioral data, and which genomes are used for comparison. There are two implementations of NCD included in the software that makes up the experiment. The first implementation is from CompLearn.org, and is a pre-release version of their planned update to their NCD library. CompLearn has decided to move away from zlib and gzip based NCD compression algorithms as they are limited in the size of data that can be compared. The new version uses LZMA compression algorithms that do not suffer from this limitation. Minor changes were made to the CompLearn source code to integrate it into the experiment software, and allow for comparisons of in-memory buffers instead of files. While perfectly functional, the LZMA based approach is computationally expensive to use. The LZMA based implementation has been left in the source code for future use, and can easily be re-enabled if needed.

With speed of the NCD algorithm being particularly critical to the extensive number of comparisons needed in the novelty search conditions of the experiment, a simple zlib based NCD algorithm was created. This zlib based class can calculate the NCD value between two in-memory data buffers, or a data buffer and a vector of comparison data buffers. The zlib approach is significantly faster than the

LZMA based approach, allowing for more comparisons to be made. A sample set of 256 data pairs were created to compare the NCD values generated by both approaches (Figure 7). While the NCD values are higher using the zlib algorithm, their relative shape remains very similar, making the compromise of using the faster zlib algorithm appropriate in order to increase the number of possible comparisons in a given amount of time.

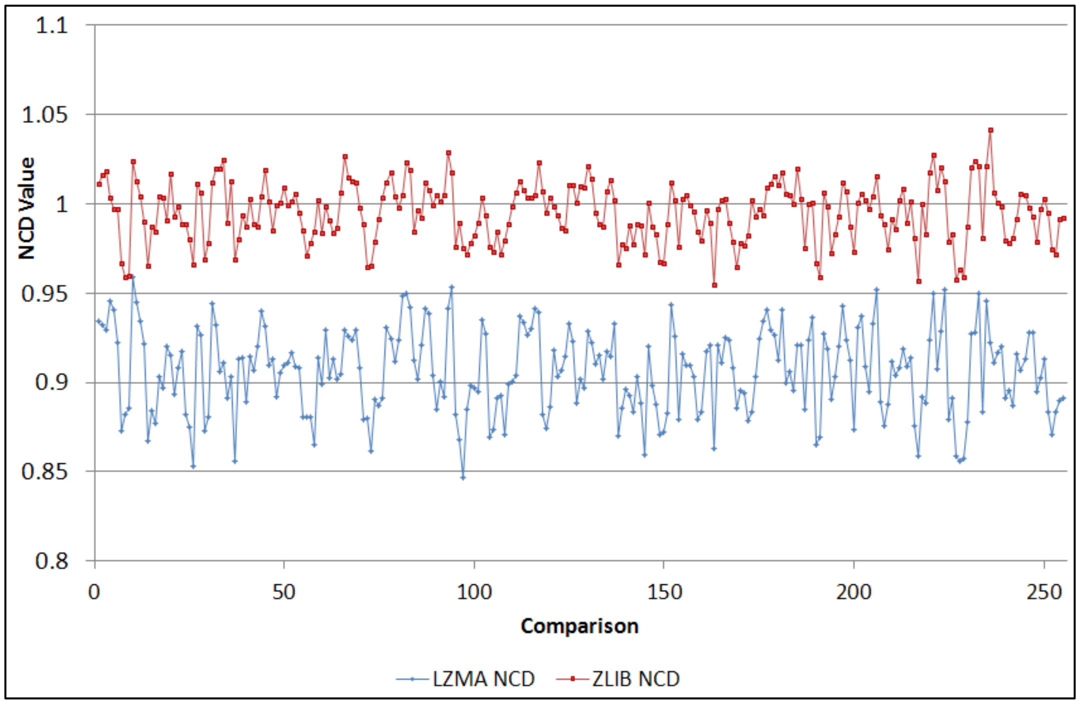

Figure 7: NCD comparisons of 256 sets of sample behavior data. The zlib algorithm gives higher NCD values for the same comparison, but the relationship of the values is very similar to that of the LZMA algorithm from CompLearn.org.

Both NCD algorithms compare in-memory char buffers. In this experiment, all of the data that goes into these buffers is serialized neural network output values, and in some conditions neural network input values as well as output values. The neural network input and outputs are double precision values in the range of -1.0 to 1.0. To simplify the data, the double values have 0.005 subtracted from them if they are negative or 0.005 added if they are positive, are multiplied by 100, and then are cast to a single byte integer. For example a double type value of 0.9573855 would be simplified to a one-byte integer of 96. Since all data being serialized into these buffers are of double type, they all undergo this conversion and simplification.

Chapter 3
Experimental Methods

Tournament Types

The experiment consists of two teams (a red team and blue team) of 256 robots each that compete against each other in the robot duel task. The experiment ends after 500 generations have been completed, although in a few of the conditions that are particularly memory intensive the computers available for running experiments would exhaust their system memory before all 500 generations were completed. Each of these teams has their own instance of the Cga genetic algorithm, so their population, speciation, and genome archives are independent of each other. There are many ways in which the competitions could be scheduled, through round-robin tournament, pareto coevolution (Noble & Watson, 2001), or an exhaustive tournament between the two teams. The experiment described here mimics the approach used by Stanley and Miikkulainen in 2004.

In each generation, every robot in a team's population is evaluated against 12 opponents. Four of the opponents are leaders (the highest ranking member) of the four highest ranking species from the other team. The final eight of the opponents are randomly selected from a hall-of-fame that contains the team champions from previous generations (Figure 8). Which hall-of-famers to use for evaluations is selected at random at the beginning of each generation, and all robots on a team compete against the same eight hall-of-famers in that generation. The robots duel on an asymmetrical game board with nine food pieces, so for each opponent the competition is repeated with the host (the robot being evaluated) and parasite robots (the opponent robots) switching starting positions, giving a total of

24 competitions to be run per generation, per robot in the population. By utilizing the leaders of the top four species of the other team, a diverse population of opponents is ensured. The inclusion of a hall-of-fame set of opponents helps to ensure that existing abilities are maintained as new ones are developed (Stanley & Miikkulainen, Competitive Coevolution through Evolutionary Complexification, 2004).

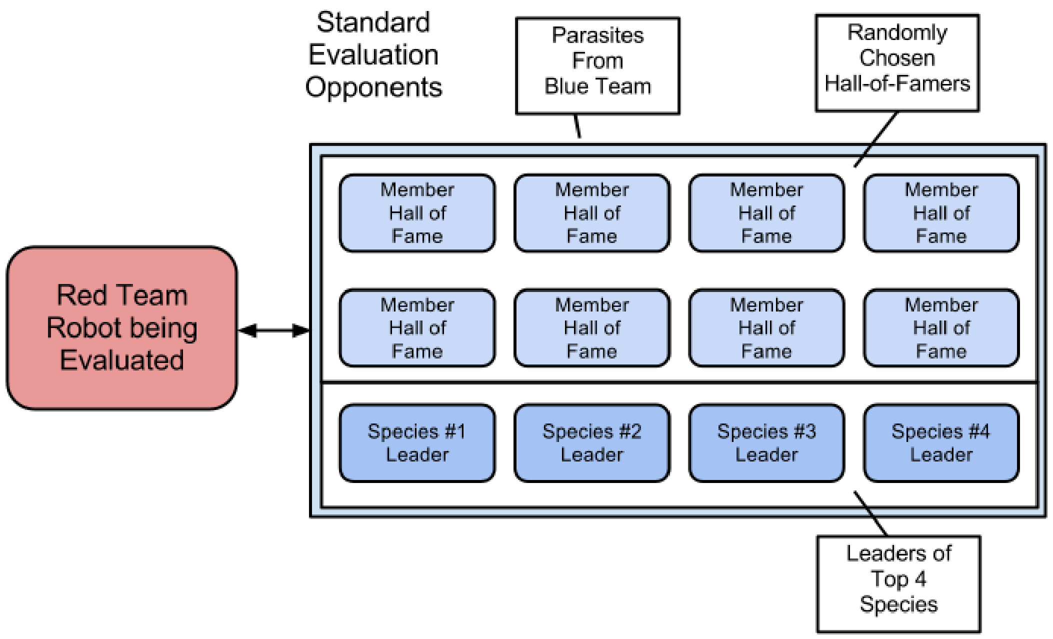

Figure 8: Example of the 12 parasites a host robot will compete against in standard evaluations. A robot on the red team will compete against parasite robots from the blue team, and vice versa. Eight of the robots are selected at random at the beginning of a generation from the other team's hall-of-fame, and the leader of the four top performing species from the other team make up the 12 parasites to be competed against. All team members play against the same 12 parasites in a given generation, so all 256 red team members would compete against these same 12 parasites in that generation. In the standard evaluation, only the host robot is being evaluated.

Matches last a maximum of 750 time steps, which corresponds with 30 seconds of real-time. If the robots have not collided within the 750 time steps, the match ends with the host robot being evaluated receiving zero points. If the parasite opponent wins the match, the host again receives zero points. The host only receives a point if it wins the match by colliding with the parasite robot while it has more energy than the parasite. The points earned by the host in its 24 matches are summed to reach its objective score for the standard evaluation. After the standard evaluation matches have been completed for both teams the robots are sorted by their scores. The highest scoring robot from each team is the team champion for that generation. If either team has multiple robots tied for first place the software moves into a playoff mode to adjust the scores of these robots tied for first place. Team champion playoffs utilize a modified version of the tournament evaluation method as opposed to the standard evaluation method.

The tournament evaluation is a direct competition between two robots, both of which are being evaluated, as opposed to a standard evaluation in which only the host robot is being evaluated. A full tournament match consists of 158 games between the two robots with 79 different food configurations, with each food configuration played twice from the two different starting positions. For the team champion playoffs, there are only 13 different food configurations used (standard nine piece food layout, and a single extra piece of food in 12 different locations) making for a total of 26 games in a playoff tournament (Figure 9). Playoffs are exhaustive in that all members of the red team tied for first place compete against all members of the blue team who are tied for first place. The number of wins, losses, and ties from these tournament matches are summed. Bonus points are added to the objective scores achieved by these robots during the standard evaluation round, with the amount of bonus points added depending on how well a robot performed in the playoff. If at the end of the playoff there is still a tie for first place after considering the wins/losses/ties in the playoff, one of the robots still tied for first place is selected randomly to be the team champion for that generation. If no robots are tied for first place at the end of the standard evaluations, the team champion playoff evaluations are skipped.

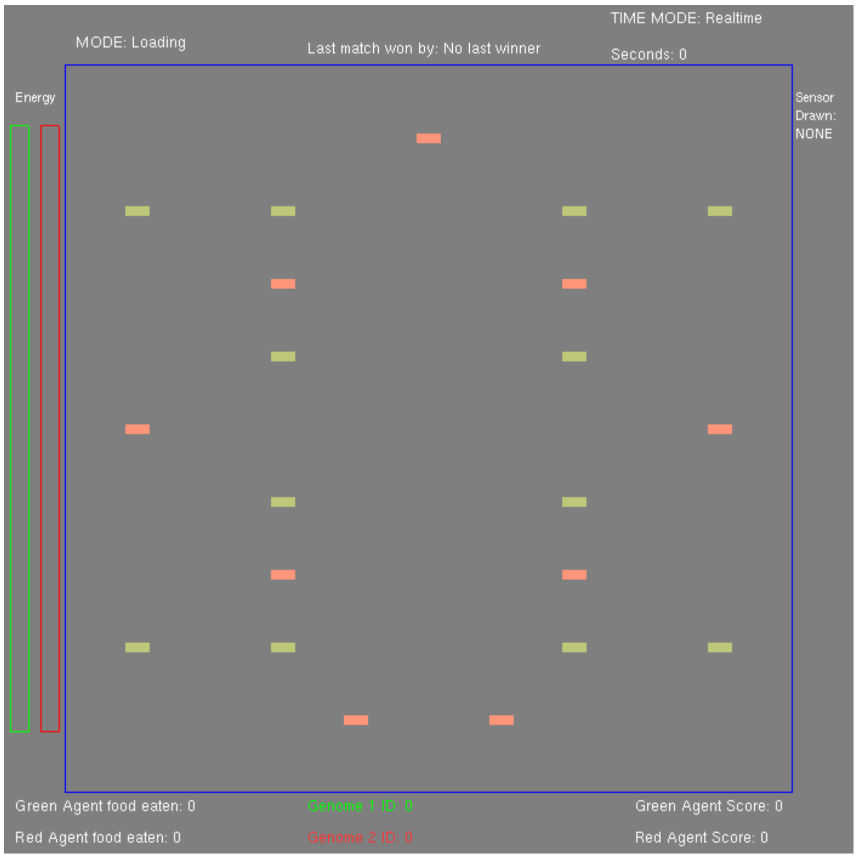

Figure 9: Game board drawing all possible positions for food. The brown rectangular food pieces are the standard nine food positions that are always used in all matches. The green food pieces shown illustrate the 12 extra food positions possible in tournament in playoff matches. No more than two of these green food positions will be used at any one time.

At the completion of a generation standard evaluation and optional team champion playoff round the team champions for both teams have been identified. In the objective NEAT condition, we now have all the information needed to evolve the next generation of neural networks. The scores achieved in the standard evaluation, plus any bonuses acquired in a team champion playoff tournament, are passed to the NEAT implementation and the next generation of neural networks are evolved. There are two more tournaments that are conducted each generation that do not affect the evolutionary system, and are undertaken simply as a means of monitoring progress in competitive coevolution. The first is that of the generation champion. The two team champions are pitted against each other in a full tournament of 158 matches with the 79 food configurations. If after the tournament the team champs are tied their scores in the standard evaluation is used as a tie breaker. If those scores are also tied, one is selected randomly to be the generation champion. The generation champion moves onto the dominance tournament.

The dominance tournament is utilized to identify a sequence of increasingly sophisticated strategies evolved in the experiment (Stanley & Miikkulainen, Competitive Coevolution through Evolutionary Complexification, 2004). In the dominance tournament match, if one robot wins more of the 158 games than its opponent, it is said to be superior to its opponent. In the first generation of the experiment, the generation champion is classified as the first dominant strategy. For each generation that follows, the generation champion plays a tournament match of 158 games against each existing dominant strategy. If it loses or ties in any of these tournament matches against an existing dominant strategy, the dominance tournament ends with no new dominant strategy found. If the generation champion defeats all existing dominant strategies, it is classified as the newest dominant strategy. This approach allows the identification of increasingly sophisticated strategies that are measurably better than previously identified strategies in an efficient manner (Stanley & Miikkulainen, The Dominance Tournament Method of Monitoring Progress in Coevolution, 2002).

Behavioral Novelty Measurement

The objective evaluations in the novelty search conditions are performed identically to the objective NEAT condition as described above, with the exception of the scores given to the genetic algorithm (Figures 10 and 11). Instead of returning the objective scores based on performance to the Cga genetic algorithm for use in evolution, a novelty score is provided in its place that represents a measure of the novelty of the behavior of a robot. By using a general approach to measuring information distance as provided by the NCD algorithms, and comparing data that encompasses the behavior of the robot, the assumption is that the NCD scores calculated are a measure of behavioral novelty. The same standard evaluations, playoffs, generation champion tournament, and dominance tournament are performed in order to allow for direct comparison of condition performance. The novelty search conditions add an additional measurement on top of the standard objective evaluations in order to arrive at these novelty scores. Four different approaches for implementing novelty search scoring were created, all using NCD as the underlying algorithm. These approaches differ in what data is compared by the NCD algorithm as well as what genomes they are compared against. These four novelty search approaches fall into two different types of behavioral sampling, dynamic match sampling, and static stimulation. All approaches use a combination of neural network inputs and outputs, or just the neural network outputs, simplified and cast to 1-byte integers, to make up the data compared using NCD.

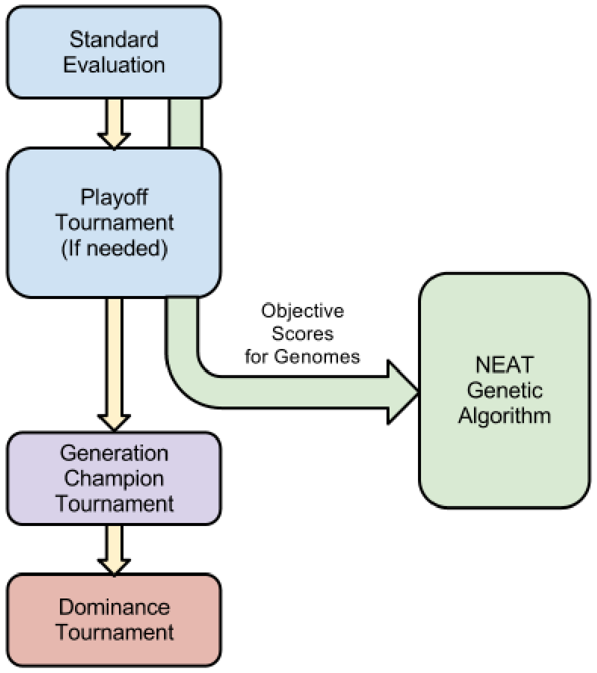
Standard Evaluation
Playoff Tournament (If needed)
Objective Scores for Genomes
NEAT Genetic Algorithm
Generation Champion Tournament
Dominance Tournament

Figure 10: Evaluation flow in the Objective NEAT condition, and where the feedback to the genetic algorithm takes place in the process of a generation.

Dynamic match sampling involves recording the inputs and outputs of a neural network during the matches in the standard evaluations. With 24 games played by each robot during standard evaluations, a behavioral record for that robot will involve the neural inputs and outputs of that robot for each match played. An important aspect to this approach is that all team members will be playing against the same 12 opponents. The 24 records of input/output data from a match remain separated so that comparisons between robots can be made with the assumption that the behavioral data being compared is from matches against the same opponent. One approach to speed the use of such methods is to only sample the first few seconds of a match to compromise between the ability to discriminate behaviors, and reduce computation time (Doncieux & Mouret, 2010). The software described in this thesis contains a parameter to control how many time steps are used as a sample to support such a compromise, however for all dynamic match sampling conditions, all 750 time steps (if the match did not end early) were used in this experiment.

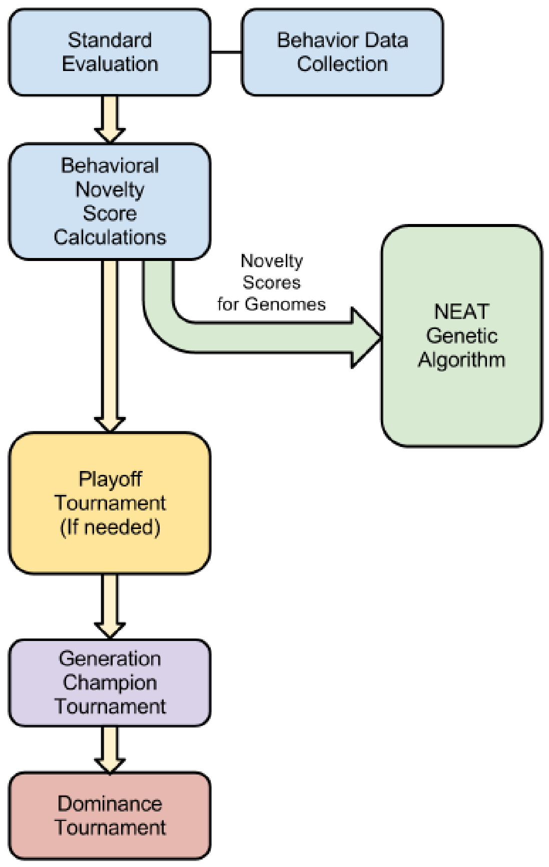
Standard Evaluation
Behavior Data Collection
Behavioral Novelty Score Calculations
Novelty Scores for Genomes
NEAT Genetic Algorithm
Playoff Tournament (If needed)
Generation Champion Tournament
Dominance Tournament

Figure 11: Evaluation flow in the Novelty Search based conditions, and where the feedback to the genetic algorithm takes place in the process of a generation.

Two novelty search approaches use the dynamic match sampling approach, one using both neural network inputs and outputs as behavior data, and another only using the neural network outputs. The conditions that used the dynamic match sampling approach took by far the longest to run, and required a number of compromises in order to reduce execution time to a reasonable length. Calculating the novelty score involves comparing the recorded behavior data to that of the other robots on the team, as well as to an archive of robots that showed particularly novel behavior in previous generations. This behavioral archive operates similarly to the hall-of-fame mechanism used in objective NEAT. Since the 24 games only involve 12 opponents, the first compromise is that for these conditions only the “left start” matches were used for calculating NCD, halving the number of comparisons. The assumption on which this decision was based on is that the behavior of a robot should be similar against an opponent regardless of which side it started on in the game. The second compromise is that only subsets of the robots on the team are compared against, instead of the entire team. In these conditions, the robots 12 records of behavior (left start only) were compared against the 12 records of behavior for 20 fellow teammates. A comparison between two robots involved 12 NCD calculations, with a comparison against 20 robot teammates, giving 240 NCD calculations before the behavioral archive comparison step of novelty score calculation.

The archived behaviors are the highest novelty scoring robot genomes from previous generations. A maximum of five new archive members were allowed each generation, and their novelty score had to be over a value of 1.0 for inclusion to the behavior archive. With such a low inclusion cutoff, in practice the top five novel robot genomes were added to the behavior archive each generation. Each robot in the population was compared against every robot in the behavioral archive for its team in addition to the 20 teammates. The NCD values calculated for each of these individual comparisons were averaged to arrive at the novelty score provided to the genetic algorithm. A complication is the underlying assumption that when the behavior data of a robot is compared to that of another robot, they were competing against the same opponent. Since robots in the behavior archive are from previous generations, this would not be true if their behavior data was saved when they were entered into the archive. To account for this problem, the genomes of the robots in the behavior archive are saved, and in each generation in which dynamic match sampling is used, these saved genomes in the behavior archive are matched against the same parasite opponents as the current generation as the general population to ensure all behavior comparisons are against the same opponents. As the generations progress and the behavior archive expands, the time and memory needed to calculate NCD scores continually increase in the conditions that use this approach.

The second approach to implementing novelty search involved the use of static stimulation instead of dynamic match sampling. Instead of recording neural network inputs and outputs from actual matches, a static series of neural network inputs are used, and the neural network outputs in response to this series of inputs are recorded as representative of behavior data. This approach is more involved in that the set of inputs has to be carefully chosen in order to represent what the robot might perceive during matches, however the speed of execution compared to dynamic match sampling is dramatically increased as comparisons go from 12 to two when comparing two robots. There are two comparisons in static stimulation in this experiment due to the size of the static stimulus set leading to behavior data that exceeds the size restrictions of zlib based NCD, so the stimulus set was split into two. The two NCD values from these static stimulation comparisons are averaged in order obtain the NCD score between two robots. There are 1,250 neural network inputs sets in the static stimulations used in these conditions. Compromises had to be made in order to keep the size of the stimulus set small enough to execute quickly, yet be representative of what the robot will likely encounter. Five different values were used for distance sensors, 0.2, 0.4, 0.6, 0.8, and 1.0. Two different values were used for the energy sensor, -0.2 and 0.2. The wall sensor was kept at 0.0. Only one food sensor would be stimulated at a time, with this approach only presenting a single food piece to the robot as opposed to what it would experience in the actual matches. All combinations of these values were used for the five enemy sensors, the five food sensors, and the single energy sensor giving a set of 1,250 input samples.

Instead of recording behavior data from standard evaluation matches, in the conditions that utilized static stimulation the neural network of the robot would be stimulated with this set of inputs, and the outputs from the neural network recorded as the behavior data to be simplified and serialized for use with the NCD algorithm. This also simplified the behavior archive handling, as all behavior data recorded is based on the same set of inputs, and retrieving the genomes of those neural networks in the behavior archive to compete against the parasite opponents of the current generation was not necessary. Similarly to the dynamic match sampling approaches, the behavior data of a robot is compared against its fellow teammates (20 in some conditions, all 255 of them in others) and the entire behavior archive. The same parameters for inclusion into the behavior archive were kept, a maximum of five new members per generation, and at least a novelty score of 1.0.

The two approaches described above make up four of the conditions. These four conditions are repeated with one minor change to make up the eight novelty search conditions. These four approaches are repeated using a minimal criteria. The implementation of minimal criteria is simple in this experiment. In the conditions using minimal criteria, the novelty scores are calculated normally as per the approach used in the condition, however if the robot did not win at least one game out of its 24 in the standard evaluation, its novelty score was assigned zero, regardless of how novel its behavior may have been. The addition of minimal criteria can help search through vast behavioral spaces more effectively (Lehman & Stanley, Revising the Evolutionary Computation Abstraction: Minimal Criteria Novelty Search, 2010). The various tests that each experimental condition undergoes can be seen in Table 1.

Experimental Condition	Standard Evaluation, Playoff, Generation Champ, and Dominance Tournament	Static Stimulation or Dynamic Match Sampling of Neural Network	Sample Neural Network Outputs Only or both Inputs and Outputs to Represent Behavior	Behavioral Archive Recompete against Current Generation Parasites	Partial or Full Team Behavioral Novelty Comparison	Full Behavioral Archive Novelty Comparison
Objective NEAT	Yes	N/A	N/A	N/A	N/A	N/A
Novelty Search	Yes	Dynamic Sampling	Inputs and Outputs	Yes	Partial	Yes
MC Novelty Search	Yes	Dynamic Sampling	Inputs and Outputs	Yes	Partial	Yes
Novelty Outputs Search	Yes	Dynamic Sampling	Outputs Only	Yes	Partial	Yes
MC Novelty Outputs Search	Yes	Dynamic Sampling	Outputs Only	Yes	Partial	Yes
Novelty Static Stimulation Search	Yes	Static Stimulation	Outputs Only	No	Partial	Yes
MC Novelty Static Stimulation Search	Yes	Static Stimulation	Outputs Only	No	Partial	Yes
Novelty Static Stimulation Extended Comparison Search	Yes	Static Stimulation	Outputs Only	No	Full	Yes
MC Novelty Static Stimulation Extended Comparison Search	Yes	Static Stimulation	Outputs Only	No	Full	Yes
Random Search	Yes	N/A	N/A	N/A	N/A	N/A

Table 1: Matrix of experimental conditions and the types of evaluations and behavioral sampling they undergo. Minimal Criteria conditions are shows with “MC” in the condition name.

Experimental Conditions

Objective NEAT

The Objective NEAT condition uses the objective scoring to evolve the genomes and is meant to represent the current approach to implementing competitive coevolution using NEAT algorithms. This condition of the experiment was run 18 times.

Novelty Search

The Novelty Search condition uses the dynamic match sampling approach, and uses both neural network inputs and outputs to represent behavior data. Behavior comparisons are made against 20 teammates, and the entire behavioral archive. This condition was run 10 times, due to the extended amount of time required to run it.

Minimal Criteria Novelty Search

The Minimal Criteria Novelty Search condition is run exactly as the Novelty Search condition, but with the addition of the minimal criteria of winning at least one match in order to receive a novelty score. This condition was run 10 times, due to the extended amount of time required to run it.

Novelty Outputs Search

The Novelty Outputs Search condition is run exactly as the Novelty Search condition, but with only the neural network outputs used to represent behavior data. This condition was run 10 times, due to the extended amount of time required to run it.

Minimal Criteria Novelty Outputs Search

The Minimal Criteria Novelty Outputs Search condition is run exactly as the Novelty Outputs Search condition, but with the addition of the minimal criteria of winning at least one match in order to receive a novelty score. This condition was run 10 times, due to the extended amount of time required to run it.

Novelty Static Stimulation Search

The Novelty Static Stimulation Search condition uses the static stimulation approach, and uses only neural network outputs to represent behavior data. Behavior comparisons are made against 20 teammates, and the entire behavioral archive. This condition was run 18 times.

Minimal Criteria Novelty Static Stimulation Search

The Minimal Criteria Novelty Static Stimulation Search condition is run exactly as the Novelty Static Stimulation Search condition, but with the addition of the minimal criteria of winning at least one match in order to receive a novelty score. This condition was run 18 times.

Novelty Static Stimulation Extended Comparison Search

The Novelty Static Stimulation Extended Comparison Search condition is run exactly as Novelty Static Stimulation Search condition is, but instead of comparing behavior against 20 teammates, all 255 teammates are compared against. This condition was run 18 times.

Minimal Criteria Novelty Static Stimulation Extended Comparison Search

The Minimal Criteria Novelty Static Stimulation Extended Comparison Search condition is run exactly as the Novelty Static Stimulation Extended Comparison Search condition, but with the addition of the minimal criteria of winning at least one match in order to receive a novelty score. This condition was run 18 times.

Random Search

The Random Search condition was added as a baseline for comparison purposes. It was run exactly as the Objective NEAT condition, with the exception that a random score between 1.0 and 2.0 was provided to the NEAT genetic algorithm as scores for the genomes.

NEAT Parameters and Measures

There are a number of parameters that influence the behavior of the NEAT genetic algorithm (Table 2). Each team had 256 members, and the experiment had a maximum of 500 generations. There was a maximum of 750 time steps in an individual match. Neural networks contained 13 inputs, and three outputs. The worst performing species of age 30 generations or older was killed off. To help protect new innovations, species less than 10 generations old received a fitness bonus multiplier of 1.3, while species older than 10 generations received a fitness penalty multiplier of 0.7. A crossover rate of 0.7 was used. The chance of adding a node was 0.025, the chance of adding a connection was 0.25, and the chance of adding a recurrent connection was 0.05. The mutation rate was 0.3, the probability a weight was replaced was 0.1, and the maximum weight perturbation was 0.5. The activation mutation rate was 0.3, and the maximum activation perturbation was 0.5. For the genome compatibility calculations, the C1, C2, and C3 values were 1.0, 1.0, and 2.0 respectively. A target number of 10 species was used, with dynamic species compatibility threshold that adjusted to meet the 10 species target.

Number of Team Members	256
Maximum Number of Generations	500
Maximum Time Steps Per Match	750
Neural Network Inputs	13
Neural Network Outputs	3
Worst Performing Species Killoff Age (Generations)	30
Young/Old Species Bonus/Penalty Generation Age Threshold (Generations)	10
Young Species Fitness Bonus	1.3
Old Species Fitness Penalty	0.7
Crossover Rate	0.7
Chance of Adding Node	0.025
Chance of Adding Connection	0.25
Chance of Adding Recurrent Connection	0.05
Mutation Rate	0.3
Probably of Weight Replacement	0.1
Maximum Weight Perturbation	0.5
Activation Mutation Rate	0.3
Maximum Activation Perturbation	0.5
C1	1.00
C2	1.00
C3	2.00
Target Number of Species	10

Table 2: Listing of the essential parameters that were used in the experiment.

The objective NEAT implementation used in this experiment did not exhibit the same level of dominant strategy transitions and complexification as seen in the original competitive coevolution experiment by Stanley and Miikkulainen. This is likely due to non-optimal parameters, and the use of a simpler NEAT implementation. This was determined to be satisfactory since the same parameters and NEAT implementation were used in all conditions of this experiment. The parameters used were validated using the built in XOR test, with 100 XOR test runs used. In the 148 experimental runs data was collected so that the number of dominant strategies found and the level of complexification of neural networks evolved could be analyzed, and the dominant strategy genomes were saved for direct competition between experimental conditions using the Genome Tester software.

Chapter 4
Results

Condition Run-Time

All conditions in the experiment were run at least once on a common computer system in order to allow for comparison of condition run-times. The fastest computer available for running experiments, with eight gigabytes of system memory and eight 3.2 GHz processor cores, was used as the common computer system between all ten conditions. There were significant differences in the amount of time required to run 500 generations depending on the experimental condition (Table 3).

Condition	**Mean Run time HH:MM:SS**	**Mean Generations Completed**	**Number of Runs on Common Computer**
Objective NEAT	0:57:25	500	2
Random Search	1:10:26	500	1
Novelty Static Stimulus	1:13:57	500	1
MC Novelty Static Stimulus	1:19:54	500	1
MC Novelty Static Stimulus EC	1:50:31	500	1
Novelty Static Stimulus EC	1:54:33	500	2
MC Novelty Outputs	3:30:43	500	1
Novelty Outputs	3:59:20	500	2
MC Novelty	6:17:01	443.8	5
Novelty	9:21:36	456	3

Table 3: Run times of the conditions on the eight processor computer available for experimental runs. Conditions using Minimal Criteria are listed with “MC”, and static stimulation conditions that compared robot behavior against all 255 teammates in the “Extended Comparison” conditions are labeled with “EC”. Both the Minimal Criteria Novelty Search and Novelty Search were not able to complete all 500 generations before the system memory was exhausted.

Dividing the evaluations and NCD calculations in the experiment across all of the processors available in the computers used for evaluations kept the time required to execute experimental runs relatively short. As expected the objective NEAT and random search conditions took the least amount of time. All eight of the novelty search conditions performed the same calculations as the objective and random conditions, plus the addition of the behavioral novelty calculations using NCD. The novelty search conditions that utilized static stimulation appear to be by far the fastest approaches to novelty search used in the experiment as they require fewer behavior comparisons, with the extended comparison (EC) versions of static stimulation taking longer to run. The dynamic match sampling conditions took the longest, with a large difference between using both the neural network inputs and outputs (Novelty and MC Novelty), or just the outputs (Novelty Outputs and MC Novelty Outputs) in the behavior data. Interestingly, in the MC Novelty and Novelty conditions, the inclusion of Minimal Criteria in the MC Novelty condition made a significant impact on the run time. These two conditions also proved to take the longest to run, even though they were only partial runs due to running out of system memory before all 500 generations had been completed. Not only were they computationally intensive, they require significantly more memory to utilize.

Number of Dominant Strategies

Condition	Mean Number of Dominant Strategies	Std. Dev.
Objective Neat	9.44	5.02
MC Novelty Stat Stimulus EC	8.22	2.10
MC Novelty Outputs	7.90	1.79
MC Novelty Static Stimulus	7.89	2.14
MC Novelty	7.40	2.41
Novelty	7.40	2.80
Novelty Static Stimulus EC	6.67	2.11
Novelty Static Stimulus	6.61	1.75
Novelty Outputs	6.20	1.62
Random	5.67	1.41

Table 4: Table of mean number of dominant strategies found in all runs of each condition. Conditions using Minimal Criteria are listed with “MC”, and static stimulation conditions that compared robot behavior against all 255 teammates in the “Extended Comparison” conditions are labeled with “EC”.

The number of dominant strategies found in an experimental run is a good measure of the condition’s ability to continually evolve ever more effective strategies that are better than the strategies evolved in earlier generations. In any given experimental run, the last dominant strategy found is the best objectively performing neural network evolved, since the dominance tournament is based on objective performance in all conditions, even those implementing novelty search. The behavioral novelty scores calculated in the various novelty search based conditions are only provided to the NEAT genetic algorithm, and do not affect the generation champion tournament or dominance tournament, which are always based upon objective performance. With the dominance tournament being a measure of objective performance across all conditions, a comparison of the number of dominant strategies found in each condition is an important measure (Table 4, Figure 12).

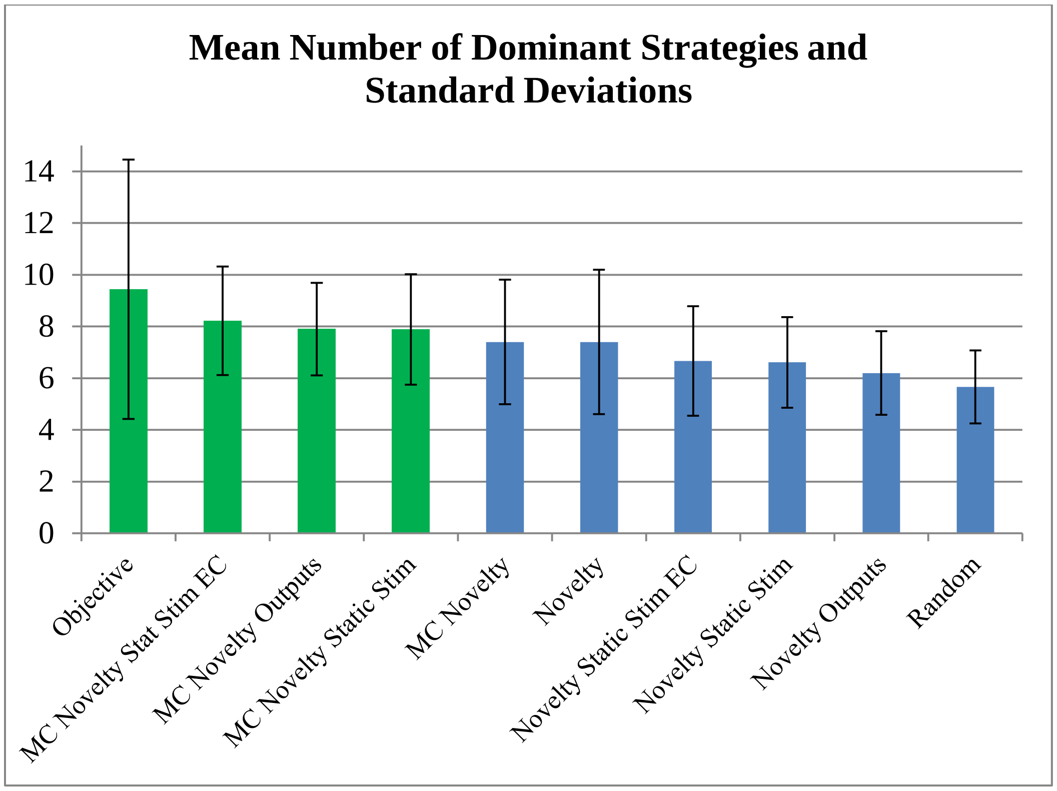

Figure 12: Mean number of dominant strategies found by each condition with standard deviation bars. The conditions shown show a statistically significant difference to the Random Search condition.

The objective NEAT condition on average found the most dominant strategies, followed by all novelty search conditions using minimal criteria. Novelty search conditions not employing minimal criteria follow, with the random search condition coming in last in mean number of dominant strategies found. In order to test the statistical significance of these differences, the number of dominant strategies found in each run of a condition was tested against every other condition using two sample t-tests assuming unequal variance (Table 5).

Comparisons	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1- Objective NEAT									
2- MC Novelty Static Stim EC	0.3506								
3- MC Novelty Outputs	0.2512	0.6729							
4- MC Novelty Static Stim	0.2388	0.6402	0.9884						
5- MC Novelty	0.1584	0.3787	0.6056	0.5999					
6- Novelty	0.1781	0.4300	0.6409	0.6380	1.0000				
7- Novelty Static Stim EC	0.0411	0.0337	0.1171	0.0938	0.4321	0.4812			
8- Novelty Static Stim	0.0345	0.0177	0.0827	0.0585	0.3787	0.4336	0.9321		
9- Novelty Outputs	0.0197	**0.0093**	0.0390	0.0277	0.2101	0.2599	0.5202	0.5392	
10- Random	**0.0060**	**0.0002**	**0.0040**	**0.0010**	0.0577	0.0916	0.1057	0.0845	0.3949

Table 5: Two-tailed P values from t-tests comparing number of dominant strategies found by each condition compared against every other condition. The numbers atop the columns correspond to the numbers by the conditions names in the first column. Values highlighted in blue indicate statistically significant differences at α=0.01. Objective NEAT, MC Novelty Static Stimulation with extended comparisons, MC Novelty Outputs, and MC Novelty Static Stimulation all show significant differences to random search.

Neural Network Complexity

The complexity of evolved neural networks was also compared between conditions. The complexity of a neural network is indicated in the number of hidden nodes (neurons that are added between the input and output neurons), and the number of connections between nodes. As the complexity of a neural network increases, it has an increased potential for more sophisticated behavior, although this is not a measure of objective performance. A simpler neural network can outperform a more complex one objectively. In order to measure the level of complexity in individual conditions, averages were taken of the number of hidden nodes and number of connections in the last dominant strategy found in all runs of a condition in generations where at least one run found a new dominant strategy.

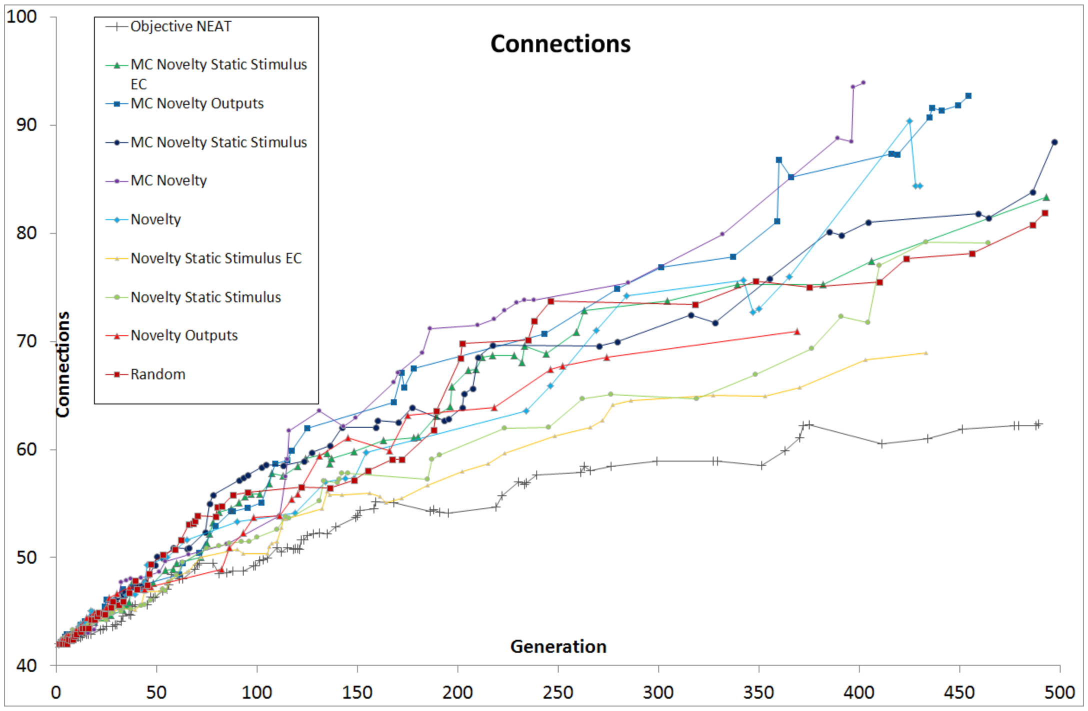

Figure 13: All conditions shown with the average number of connections in the best dominant strategy found in each condition run at generations in which one of the runs found a new dominant strategy.

Interestingly, all conditions show greater complexity in their neural networks than the objective NEAT condition, although this does not translate into better objective performance as will be shown shortly (Figures 13, 14). Figures 15 and 16 show a de-cluttered version of these graphs with only the Objective NEAT, Random Search, and the MC Novelty Static Stimulation with Extended Comparisons, MC Novelty Static Stimulation, and MC Novelty Outputs conditions, which are the three novelty search conditions which show significant differences from random search in number of dominant strategies.

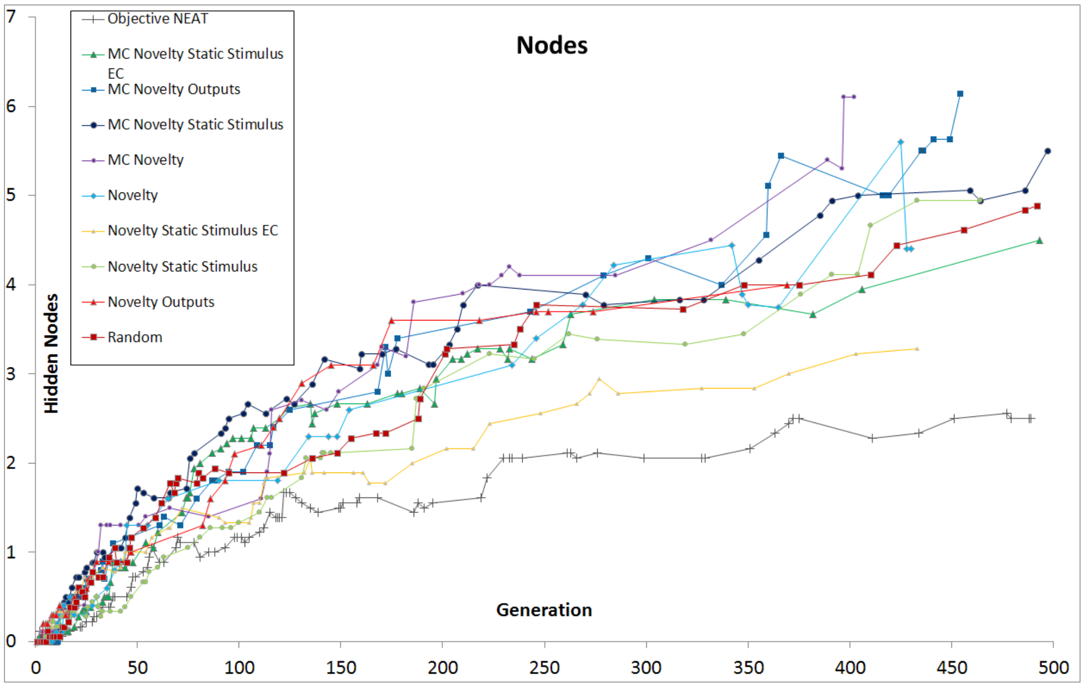

Figure 14: All conditions shown with the average number of hidden nodes in the best dominant strategy found in each condition run at generations in which one of the runs found a new dominant strategy.

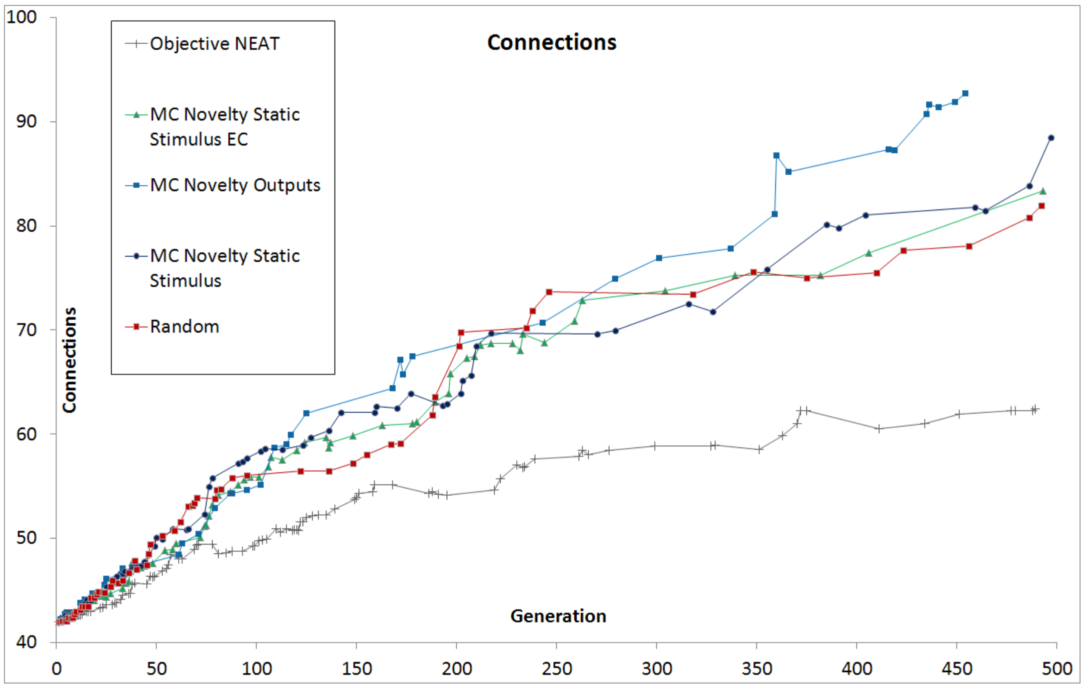

Figure 35: Identical to figure 13, but with fewer conditions shown.

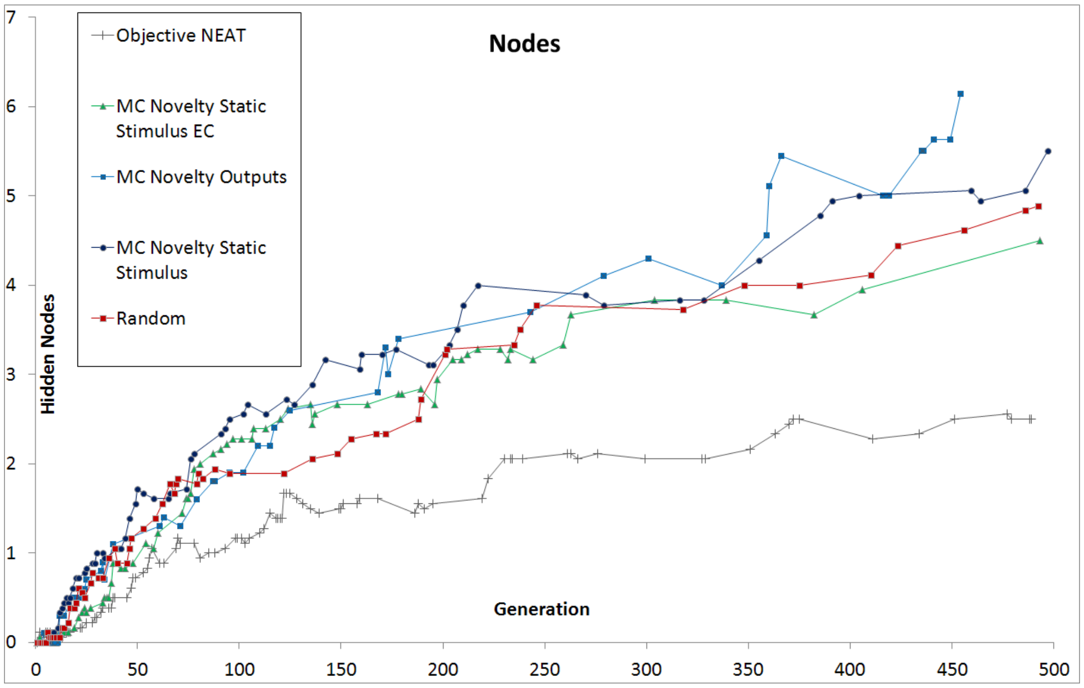

Figure 16: Identical to figure 14, but with fewer conditions shown.

<u>Round-Robin Tournament Between Conditions</u>

While the novelty search conditions do not use objective performance in the genetic algorithm and evolutionary process, ultimately in this particular problem domain (as in many others) the goal is to evolve neural networks with higher objective performance. The dominance tournament results seen previously help to measure the capability of individual conditions for evolving neural networks that exhibit ever more capable neural networks in objective performance. With the genomes of all dominant strategies found being saved during the experimental runs, it is possible to have the dominant strategies from all conditions compete directly against each other using the Genome Tester software. With six conditions having

18 runs each, and four conditions having 10 runs, we have 148 players (or dominant strategies) for an exhaustive round-robin tournament. Due to network file server errors during the experimental runs, four of these genomes were not saved correctly and could not be used in the round-robin tournament. Genomes are missing for the dominant strategies of two runs of the Minimal Criteria Novelty Static Stimulation with Extended Comparisons condition, one run of the Novelty Static Stimulation with Extended Comparisons condition, and from one run of the Random Search condition, leaving a total of 144 players to be directly compared against each other in objective performance.

The Genome Tester software read in the list of 144 genomes, and used them to create neural networks to drive the robots in the competition. Each of these robots played every other robot in the list for an exhaustive round-robin tournament. Each game between two robots encompassed the full tournament type used in the experiment, with 79 different food layouts and 158 games. Wins, losses, and ties were tracked between robots, so if a robot won the majority of the 158 games against a single robot, it was given a single win. With 144 players, 10,296 robot competitions were played in total, with 158 games in each competition, for 1,626,768 simulated games. The Genome Tester ran these competitions in a single thread, and took roughly two and a half hours to complete the evaluations on a 3.0 GHz processor, without rendering the matches as they ran at accelerated rates. Looking at the mean number of wins by dominant strategies from each condition (Table 6), once again the Objective NEAT condition performed the best by objective measures, Random Search is the worst, with the various novelty search condition falling in between the two on objective performance measurements. Within the novelty search conditions, the Minimal Criteria based conditions once again performed better than the non-minimal criteria based conditions.

Condition	Mean Number of Wins	Std. Dev.
Objective NEAT	92.22	28.17
MC Novelty Outputs	76.30	18.55
MC Novelty	67.50	30.05
MC Novelty Static Stimulus EC	63.44	22.35
MC Novelty Static Stimulus	60.50	15.83
Novelty Static Stimulus	60.39	17.73
Novelty	54.90	16.88
Novelty Static Stimulus EC	54.47	13.30
Novelty Outputs	52.30	15.54
Random	40.12	18.30

Table 6: Mean number of wins of dominant strategies from each condition, along with standard deviation.

The statistics on mean novelty scores for the conditions that used minimal criteria were skewed by the zeroing of scores of the robots that did not pass the minimal criteria, however statistics on mean novelty scores from non-minimal criteria conditions are interesting. The range of novelty scores as calculated using NCD was exceptionally small. The mean and standard deviation of novelty scores for each team in each generation of the non-minimal criteria conditions were averaged. The overall mean novelty score was 1.08, with a standard deviation of 0.03. The minimum average novelty score for a single team in a single generation from this data set was 0.98, and the maximum 1.17. NCD algorithms tend to give measurements in the range of zero to one; however the zlib based NCD algorithms skew a bit higher in general. The small range of NCD values that made up the novelty scores is surprising.

Neural Network Visualization

The neural network for the overall winner of the round-robin tournament can be seen in figure 17. This neural network is from the objective NEAT condition and won 125 games, lost 18, and had zero ties. Figure 18 shows the best neural network from the Minimal Criteria Novelty Outputs condition which had 106 wins, 31 losses, and six ties. These renderings are captured from the Genome Tester software. Circular links connecting back on the same neuron, as well as dashed white lines between two neurons represent recurrent connections.

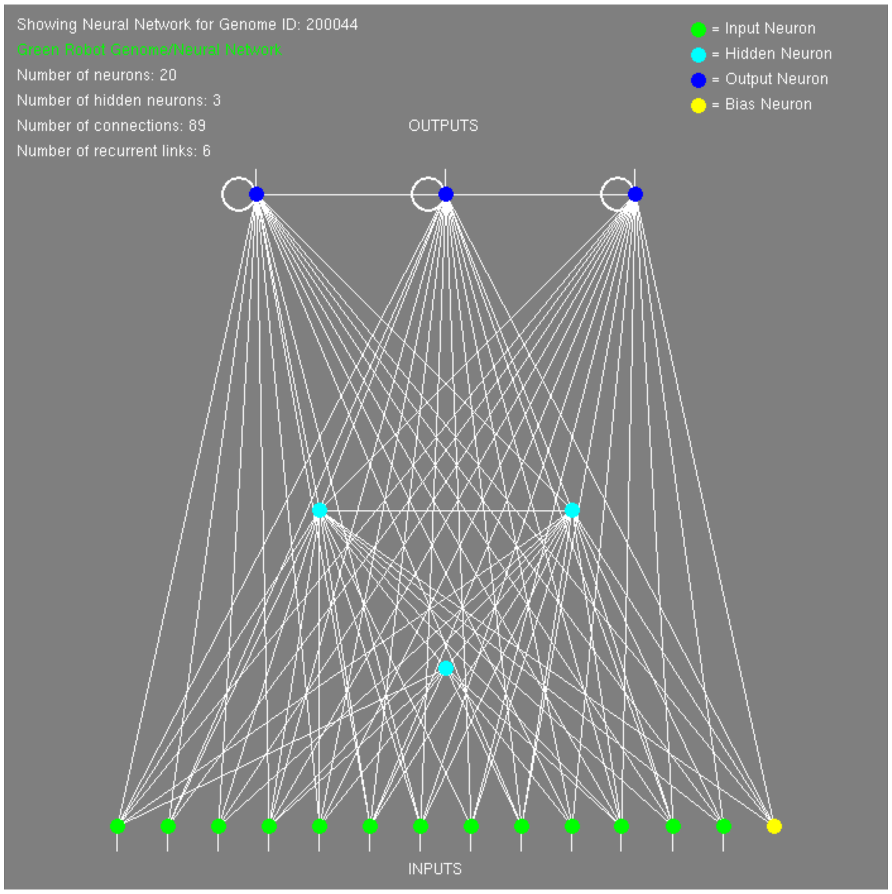
Showing Neural Network for Genome ID: 200044
Green Robot Genome/Neural Network
Number of neurons: 20
Number of hidden neurons: 3
Number of connections: 89
Number of recurrent links: 6
OUTPUTS
● = Input Neuron
● = Hidden Neuron
● = Output Neuron
● = Bias Neuron
INPUTS

Figure 17: Neural network that won the round-robin tournament. This neural network was evolved using the objective NEAT method. This particular neural network has three hidden nodes, 89 connections, and six recurrent connections.

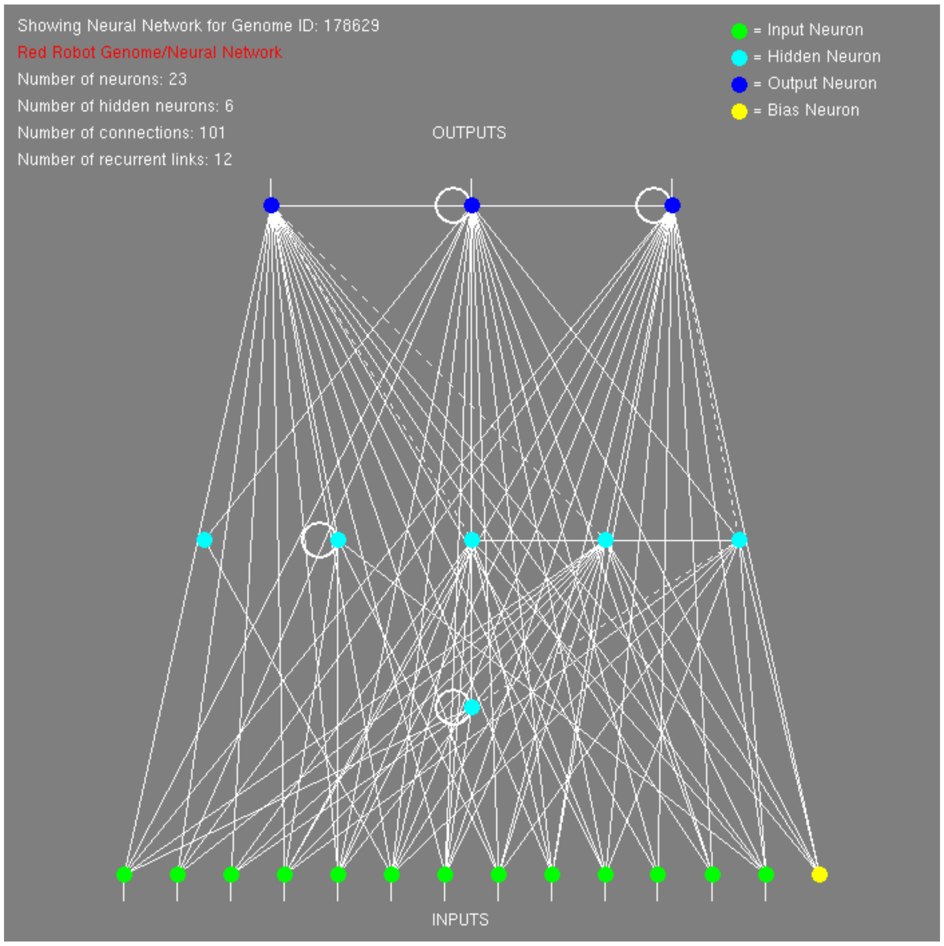
Showing Neural Network for Genome ID: 178629
Red Robot Genome/Neural Network
Number of neurons: 23
Number of hidden neurons: 6
Number of connections: 101
Number of recurrent links: 12

Figure 18: Best performing neural network from the Minimal Criteria Novelty Outputs condition. This particular neural network has six hidden nodes, 101 connections, and 12 recurrent connections.

Chapter 5
Discussion and Conclusions

This experiment represents the first time that novelty search has been implemented in a competitive coevolutionary problem domain using NEAT genetic algorithms. The results of this study indicate that the objective NEAT approach is more effective than any of the novelty search approaches for this particular application. The hypothesis that NCD based novelty search works in competitive coevolutionary problem domains is, however, supported by the results of the experiment, even though these novelty search implementations did not appear to perform better than the existing objective NEAT approach in objective performance. It is quite possible in a competitive problem domain that exhibits higher levels of deception that the novelty search conditions would outperform objective NEAT in objective measures, as the objective approaches would tend to become trapped in local optima. Regardless of the objective performance of novelty search in this experiment, the possibility of a working novelty search implementation for competitive coevolutionary problems is useful. Novelty search is limited by the fact that it ignores the objective completely, and has no bias towards optimization once a solution is found. Such fine optimizations are better achieved by objective NEAT, with novelty search better at finding approximate solutions (Lehman & Stanley, Exploiting Open-Endedness to Solve Problems Through the Search for Novelty, 2008). A hybrid implementation of these two approaches could very well outperform either single approach implemented on its own in many problem domains. Utilizing objective NEAT to optimize approximate solutions found by novelty search, or simply using novelty search to escape from local optima if objective NEAT becomes stuck for an extended period of time would likely be two very successful approaches to solving complex competitive coevolutionary problems. With all novelty search conditions showing higher levels of complexification than the objective approach, switching to novelty search when objective search is trapped in local optima could also lead to a boost in the topology of the neural network allowing continued successful search for more sophisticated solutions when the objective approach is re-enabled and has new neural network structure to optimize.

With such small variations in the novelty scores found by the NCD algorithms, the approaches used in this experiment could be called weak measures of behavioral novelty at best. Despite this, it is clear that some of these approaches outperform random search, and appear to be functional methods of novelty search. If the novelty scores generated by the NCD algorithms were not meaningful measurements it would be expected that the novelty search conditions would perform no better than the random search condition which was not the case. The small variation in novelty scores is concerning and likely indicative of the difficulties in applying a general method of measuring novelty in a complex problem such as competitive coevolution. The advantages of using a general approach to measuring behavioral novelty that does not require expert knowledge of the problem domain, and time consuming and potentially error prone development of problem specific behavioral metrics should help mitigate concerns of relying upon a weak measure of behavioral novelty. It would prove interesting to devise a competitive coevolutionary problem that involved high levels of deception that would provide a test to see if the NCD approaches to generating novelty scores is indeed measuring behavioral novelty, although with some of the static stimulation and output-only dynamic match sampling conditions performing well and only comparing the neural network outputs, this is likely the case since the only data being fed into the NCD algorithms is the behavioral outputs of the neural networks. Given the weak measurement nature of the NCD algorithms presented here, the most effective method of combining novelty search and objective NEAT in competitive coevolution is likely to use novelty search as a means of escaping local optima. One approach would be using the objective method as the primary means of driving evolution, but if a new dominant strategy has not been found in some number of generations, change to novelty search using NCD for a few generations or until a new dominant strategy is found.

There were eight variations of novelty search implemented in this experiment, all of which take longer to run than running traditional objective based NEAT implementations. In the case of problem domains where objective performance is important, such as the robot duel used in this experiment, the results point to the use of Minimal Criteria as being helpful, regardless of which specific novelty search method is used. Not only does the inclusion of minimal criteria appear to improve the objective performance, but it also appears to provide a benefit in computation time. Previous papers have considered the application of NCD too slow for their problem domains (Doncieux & Mouret, 2010), and certain novelty search conditions used in this experiment do show large increases in processing time when compared to objective NEAT. In particular the novelty search conditions that used both neural network inputs and outputs to represent behavior data took significantly longer to run, and proved to require large amounts of system memory. The fastest implementations of novelty search in this experiment involved the use of static stimulation. The implementations that used only neural network outputs from dynamic matches fell in the middle when using computation time as the criterion. For some problem domains, the use of NCD to create a novelty scoring metric might just be fast enough to be considered.

In regards to objective performance, three implementations of novelty search stand out, those of MC (Minimal Criteria) Novelty Static Stimulation with EC (Extended Comparisons), MC Novelty Static Stimulation without EC, and MC Novelty using neural network outputs only from dynamic match sampling. The two static stimulation approaches have relatively low run-times, while the Minimal Criteria Novelty using neural network outputs takes much more time to run. These three conditions performed well in the number of dominant strategies found, and performed well in the round-robin tournament. Which of these methods are appropriate for application depends largely on the tradeoffs of run-time and ease of implementation. The MC Novelty Outputs condition takes longer to run, but does not require the creation of a static stimulation set, which is an effort that seems somewhat closer to the development of novelty metrics using expert domain knowledge of the problem area, as opposed to a truly general approach to creation novelty metrics using NCD. Once a static stimulation set is created however, the number of behavioral comparisons is significantly reduced, management of the behavioral archive is eased, and run-times are faster. These tradeoffs mean there is no clear cut recommendation between the application of static stimulation or dynamic match sampling using neural network outputs. The use of dynamic match sampling of both input and output data seems to be a poor choice in general with its long run-times. The use of both neural network inputs and outputs in dynamic match sampling conditions likely lead to a poor measure of behavioral novelty in this particular experiment as there are 12 inputs and 3 outputs in the data representing behavior. The neural network outputs representing the actual behavior of the neural network are likely lost in the noise of the far greater number of inputs.

This paper introduced a number of methods of implementing novelty search in a competitive coevolutionary problem domain, all utilizing NCD algorithms to provide a general method of measuring behavioral novelty. While taking longer to run due to the additional computation of calculating behavioral novelty, some of the more promising approaches to novelty search proved to be the faster methods of implementing novelty search. As the number of processing cores continues to increase, and with the ease of dividing evaluations across multiple cores, the use of computationally intensive behavioral novelty metrics will increasingly become feasible in solving a wider range of competitive coevolutionary problems. The novelty search versions of the experiment did not outperform the objective based approach to evolving neural networks in the competitive coevolution problem domain; however the combination of the two approaches would likely outperform objective competitive coevolution by itself, especially as more complex competitive coevolution problems are attempted with NEAT.

Works Cited

Buckland, M. (2002). *AI Techniques for Game Programming.* Premier Press.

Cilibrasi, R., & Vitanyi, P. (2005). Clustering by Compression. *IEEE Transactions on Information Theory*, 1523-1545.

Doncieux, S., & Mouret, J. B. (2010). Behavioral Diversity Measures for Evolutionary Robotics. *IEEE Congress on Evolutionary Computation.*

Gomez, F. J. (2009). Sustaining Diversity using Behavioral Information Distance. *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 113-120). ACM Press.

Lehman, J., & Stanley, K. O. (2008). Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. *Proceedings of the Eleventh International Conference on Artificial Life.* Cambridge: MIT Press.

Lehman, J., & Stanley, K. O. (2010). Revising the Evolutionary Computation Abstraction: Minimal Criteria Novelty Search. *Proceedings of the Genetic and Evolutionary Computation Conference.* ACM Press.

Noble, J., & Watson, R. A. (2001). Pareto Coevolution: Using Performance Against Coevolved Opponents in a Game as Dimensions for Pareto Selection. *Proceedings of the Genetic and Evolutionary Computation Conference.* Morgan Kaufmann.

Rosin, C. D., & Belew, R. K. (1997). New Methods for Competitive Coevolution. *Evolutionary Computation*, vol. 5, no. 1, pp. 1-29.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 99-127.

Stanley, K. O., & Miikkulainen, R. (2002). The Dominance Tournament Method of Monitoring Progress in Coevolution. *Proceedings of the Genetic and Evolutionary Computation Conference.* Morgan Kaufmann.

Stanley, K. O., & Miikkulainen, R. (2004). Competitive Coevolution through Evolutionary Complexification. *Journal of Artificial Intelligence Research*, vol. 21, pp. 63-100.

Appendix

Hostname	CPU Type	Physical CPU Cores	Logical CPU Cores	System Memory
Beast	Intel Core2 Extreme X9775	8	8	8 GB
Nova	Intel Core i7 950	4	8	6 GB
Hercules	Intel Core i7 940	4	8	6 GB
Colossus	Intel Core i7 920	4	8	6 GB
Colossus_Left	Intel Core i7 920	4	8	6 GB
Colossus_Center	Intel Core i7 920	4	8	6 GB
Colossus_Right	Intel Core i7 920	4	8	6 GB
Colossus_TopRight	Intel Core i7 920	4	8	6 GB
Mongo	AMD Opteron 285	4	4	4 GB
Opus	Intel Core i5 2300	4	4	8 GB

Specifications of the computers used to run experiments on nights and weekends. All experimental conditions were run at least once on “Beast” for comparison of condition run-times.

Requests for source code or for copies of data should be sent to the author Drew Kirkpatrick at: drew.kirkpatrick@gmail.com