Project Report

Cai Crumley

October 18, 2013

1 Introduction

This project conducted research in the field of neuroevolution (NE), in which evolutionary algorithms (EAs) are applied to generate artificial neural networks (ANNs). In particular, we attempted to improve upon the NE method called Multi-Agent Enforced Sub-Populations (ESP) by using behavioural and genotype metrics to control sharing genetic material between agents. We used a simulation of a simple predato-prey scenario as the testbed to compare Multi-Agent ESP and its modifications with varying parameters.

An ANN is a processing structure that simulates a neural network. An ANN consists of a set of nodes or neurons with weighted, directed connections between them. ANNs can be classified as either feedforward or recurrant depending on the connections between the nodes. An ANN is classified as feedforward if there exists a way of numbering the nodes such that there are no connections from a node to another node with a lower number. If no such numbering exists then the ANN is classified as recurrant [25]. Thus a feedforward ANN can be modelled by a directed acyclic graph, whereas a recurrant ANN cannot. The architecture of feedforward networks typically consists of layers of nodes in which each node is connected to all nodes in adjacent layers, so that the outputs of a layer of nodes become the inputs to the next layer. The first layer of such an ANN corresponds to external inputs and the last layer corresponds to the outputs of the ANN. The output of a node is a function of the sum of its inputs minus the bias, or threshhold, of the node. Formally, we can express this using the following equation [25]:

$$y_i = f_i \left(\sum_{j=1}^{n} w_{ij} x_j - \theta_i \right)$$

where y_i is the output of node i, n is the number of incoming connections, x_j is the jth input to node i, w_{ij} is the weight of the connection between node i and the node with the output correponding to x_j, and θ_i is the bias or threshhold of

node i. f_i is called the transfer function and is typically a nonlinear function such as a Gaussian or sigmoid function [25].

Evolutionary algorithms are population-based stochastic search algorithms that mimic evolution in nature in order to generate a solution to a problem [5][25]. A population of genotypes that represent candidate solutions to the problem is gradually improved by repeated application of genetic operators, including crossover, mutation and selection. The evolution process is guided by a fitness function which defines the goal of the algorithm and is used to rate the fitness of the genotypes. In each generation genotypes are evaluated using the fitness function, and then genotypes that have a poor fitness are replaced with offspring created by applying the crossover and/or mutation operators to fitter genotypes. This process is repeated until some termination condition is met, such as a maximum number of generations being reached or a desired fitness level being achieved. EAs are well suited to solving high-dimensional optimisation problems that have complex fitness landscapes containing many local optima [25].

EAs can been used in conjunction with ANNs in a few different ways. They can be used to optimise the connection weights between nodes (which is known as training the ANN), and they can also be used to optimise the overall architecture or topology of ANNs [25]. EAs can also be used together with other algorithms as part of the traning process [25]. In this project we used a predefined architecture and evolved only the connection weights.

In this project we used a predator-prey simulation similar to the one used by Yong and Miikkulainen in [26] in which 3 predators attempt to capture a single prey on a toroidal grid-based world. This project attempts to expand on the work done by Gomez and Miikkulainen in [7] and Yong and Miikkulainen in [26]. We modify the evolution process by applying crossover between populations of genotypes of different predators based on their similarity, as measured by behavioural and genotype metrics. The hypothesis of this research is that, by using inter-agent crossover appropriately, we can reduce the number of generations taken until a predetermined level of fitness is achieved.

2 Background

2.1 Neuroevolution

EAs are well suited to searching over large, complex search spaces that have many local optima, and are less likely to get stuck in local optima than gradient-based methods [25]. Since they do not rely on gradient information, they can be used in situations where the search space is non-continuous or non-differentiable, or where gradient information is difficult or expensive to obtain. Such situations

would be problematic for gradient-based search methods. Thus EAs allow for greater flexibility in the definition of a fitness function versus an error function for a gradient-based method [5][25].

When applying EAs to find an optimal set of connection weights and/or architecture of an ANN, the decisions of how to encode the ANN in a genotype and what genetic operators to use are very important. In [5], Floreano et al. provide three clasifications of genetic representations: direct, developmental and implicit. Direct and developmental representations are discussed in more detail below. Implicit representations are inspired by gene regulatory networks in biology, and won't be examined in this report.

2.1.1 Direct Representations

Direct representations have a one-to-one mapping between the genotype and the parameters of the ANN. For example, if the network architecture is fixed, then the connection weights could simply be encoded as a binary string with each weight corresponding to a fixed number of bits, or the weights could be encoded as a vector of real numbers. However, it is also possible to specify the architecture of an ANN in a genotype using a direct representation. In [12], Igel achieved very good results using direct encoding with Covariance Matrix Adaption Evolution Strategy on pole balancing problems using realtively small recurrant ANNs with between 4 and 16 hidden nodes. Floreano et al. suggest that direct representations may not work as well for large networks as they do for small networks, since the size of the genotype must scale with the size of the network which may impede the evolution process [5]. In [25], Yao makes the same observation with regards to direct encoding of network architecture using a connection matrix: large networks will require a large matrix, which greatly increases the size of the search space. However, he points out that by constraining the network architecture, for example to a layered feedforward architecture, the length of the genotype can be greatly reduced. A similar idea could be applied to training ANNs, but it would require knowledge about the search space which is unlikely to be available, especially for large, complex search spaces associated with large ANNs.

Symbiotic, Adaptive Neuroevolution (SANE), introduced by Moriarty and Mikkulainen in [15] is an example of a direct encoding scheme in which each individual genotype in the population encodes the weights of a node in the hidden layerof an ANN. SANE and its extensions are discussed in detail below. Neuroevolution of Augmenting Topologies (NEAT) is another direct encoding scheme introduced by Stanley and Mikkulainen in [20] in which the architecture of ANNs and their connection weights are simultaneously evolved. NEAT attempts to find the minimal necessary topology of an ANN by incrementally growing individual networks. It uses uses historical markers that indicate when a gene (representing either a node

or a connection) is added to a genotype in order to allow for meaningful crossover between networks with different topologies. NEAT also protects innovation in topology by using speciation. This allows the algorithm to explore topological changes even if they initially result in reduced fitness (which is often the case). This also promotes diversity, which helps to avoid premature convergence on local optima, which is a common problem with NE [5][15].

The competing conventions problem, also known as the permutation problem, is a major problem in NE [5][20][25]. The problem arises from the fact that the same ANN has many different possible representations. For example, if two ANNs differ only in the order of nodes in the hidden layer then they are effectively the same ANN, but they would usually not have the same genetic representation. This means that two genotypes which appear dissimilar may represent the same solution, and applying crossover to such genotypes would very likely result in offspring with poor fitness. This problem is somewhat addressed in NEAT using the historical markings. By tracking the origins of genes, NEAT is able to identify which genes correspond to the same structure, and Stanley and Mikkulainen suggest that such corresponding genes are likely to have the same function. This allows for meaningful crossover between the corresponding parts of the genotypes, even if the networks are topologically dissimilar.

2.1.2 Developmental Representations

Developmental representations offer an alternative to direct representations. Whereas direct representation methods explicitly encode the architecture and/or connection weights of an ANN in the genotype, developmental representations instead use the genotype to specify a set of developmental rules which are applied to construct an ANN. This approach allows for large networks to be generated from a compact genotype [5][25].

One developmental approach is to use production rules. Each individual genotype can represent either a single rule or a set of rules [25]. Each rule specifies a symbol substitution procedure with a left-hand side and a right-hand side. Non-terminal symbols on the left-hand side of the rule are replaced by terminal or non-terminal symbols on the right-hand side. Starting from an initial state, suitable rules that match the symbols in the current state are applied until only non-terminal symbols remain. The final state then specifies the architecture and/or weights of an ANN. Kitano uses such a system in [13] to develop a connectivity matrix which specifies the topology of an ANN. His rules each turn a symbol into a 2×2 matrix. Non-terminal symbols become matrices of symbols, while terminal symbols become matrices containing 0s and 1s. The final matrix is interpreted as a connectivity matrix where a 1 in row i, colmun j specifies a directional connection between node i and node j while a 0 specifies no connection.

Cellular encoding (CE) is another developmental approach which was introduced by Gruau in [9]. In this method, genotypes consist of a set of graph transformations that are applied to turn an initial single cell into a full ANN. Each of these transformations either divides a mother cell into two daughter cells, where the particular rule determines what properties of the mother cell the daughter cells inherit, or changes the local topology of a cell, or adjusts connection weights. Graph transformations are organised into trees which are parsed during the genotype to phenotype development process. Leaf nodes of transform trees can contain pointers which lead to the root of another transform tree. This system allows for repeated sub-networks to develop in a single ANN. CE is used to generate controllers for 6-legged robots in [10], and in [11] CE is applied to the two-pole balancing problem.

2.2 SANE, ESP and Multi-Agent ESP

The NE method used in this project is based on Multi-Agent ESP, which is an extension of ESP, which is itself and extension of SANE. These methods are described in detail in this section.

2.2.1 SANE

The primary insight of SANE [15] is to use each individual in the population to encode only a partial solution to the problem, which is a method called symbiotic evolution. Traditional neuro-evolution in which each individual encodes an entire ANN often results in premature convergence on local optima, which necessitates the use of diversity encouragement through methods such as fitness sharing or crowding. However, with symbiotic evolution, diversity is automatically encouraged. Effective solutions can only be constructed out of individuals that each fulfill a specialised role in the solution, and for such specialisation to emerge the population must maintain diversity. During evaluation, an individual is used together with different combinations of other individuals in a given number of solutions. The fitness of an individual is calculated as the average fitness over all of the final solutions in which the individual participated. Moriarty and Mikkulainen assert that this evaluation process ensures that specialisations will emerge: since the fitnesses of individuals are based on final solutions, and since individuals do not specify an entire solution by themselves, individuals that specialise to solve a particular subproblem will perform the best. In symbiotic evolution, evolutionary pressure prevents the population from converging on a single specialisation, since effective solutions require different specialisations to be present. Thus populations remain diverse and unconverged, even in prolonged evolution.

SANE evolves connection weights for neurons in hidden layers for ANNs with

a fixed architecture. SANE does to need to be used with a particular architecture or genotype representation. In [15] Moriarty and Mikkulainen used a layered feedforward ANN with two hidden layers, and a bitwise genotype each of which contains a series of connection definitions for a particular node. Each connection has an 8-bit label field and a 16-bit weight field. The most significant bit in the label field determines whether the connection is incoming or outgoing. The value of the label field modulo the number of nodes in the connected layer determines the other node in the connection. For example, in a single layer ANN with 8 output nodes numbered 0 through 7, a label value of 166 would specify a connection to node 6 in the output layer. If the label value was less than 128 it would specify a connection to a node in the input layer. The 16-bit weight field encodes a floating point value which is used as the weight of the connection.

Evolution using SANE proceeds as follows: A number of neurons are randomly selected from the population equal to the number of hidden nodes in the ANN architecture being used. The selected neurons are combined to form an ANN, which is then evaluated as a solution to the problem. The fitness value of the ANN is added to each neuron's fitness score, and the number of networks in which each neuron has participated is incremented. This selection and evaluation procedure is repeated until each neuron has been used in a sufficient number of networks. The fitness score of each neuron is then divided by the number of times the neuron was used to get an average fitness of the networks that the neuron was part of. Crossover and mutation operators are then applied to the population based on their fitness scores.

Moriarty and Mikkulainen used rank-based selection, where the top 25% of the population is paired up for crossover. The lowest ranked genotypes are replaced by the offspring produced. They used a one-point crossover operator, producing two offspring per crossover. They also applied mutation to the offspring with a probability of 0.1% in order to allow genetic material that is missing from the initial population or lost via crossover to be reintroduced to the population. Rank-based selection was chosen over fitness-proportional selection so that selection would be biased towards the fittest neurons even in the case where the fitness values in the population are similar.

Moriarty and Mikkulainen compared SANE to a number of other neuroevolution methods using a single pole-balancing problem, also known as the inverted pedulum problem. The problem and similar variants are often used as benchmark problems in neuroevolution [7][12][20]. The single pole-balancing problem consists of a simulation in which a pole is balanced on top of a cart on a fixed-length track. The task of the controller is to push the cart left or right along the track with a fixed-sized force in order to keep the pole from falling while avoiding the boundaries of the track. The controller is provided input about the physical properties of

Method	Mean	Best	Worst	SD
1-layer AHC	130.6	17	3017	423.6
2-layer AHC	99.1	17	863	158.6
Q-learning	19.8	5	99	17.9
GENITOR	9.5	4	45	7.4
SANE	5.9	4	8	0.6

Table 1: CPU time in seconds required to find a successful network when starting with centered cart and pole

Method	Mean	Best	Worst	SD
1-layer AHC	49.4	14	250	52.6
2-layer AHC	83.8	13	311	61.6
Q-learning	12.2	4	41	7.8
GENITOR	9.8	4	54	7.9
SANE	5.2	4	9	1.1

Table 2: CPU time in seconds required to find a successful network when staring with random cart and pole positions and random initial velocities

the system, including the position of the cart, the velocity of the cart, the angle of the pole relative to the cart and the angular velocity of the pole. Other variants of the problem have two poles balanced on the cart instead of one, and may provide different input to the controller, such as denying it information about the velocity of the pole(s). The problem used by Moriarty and Mikkulainen is one of the easier variations, and could be too easy for many modern NE methods [20].

The methods that Moriarty and Mikkulainen compared SANE to include single-layer Adaptive Heuristic Critic (AHC) [3], two-layer AHC [1], Q-learning [21] and the GENITOR system [22]. The results of their experiments are summarised in tables 1 and 2. Each simulation was run 50 times. The results indicate that SANE was able to find a successful controller significantly faster than all of the other methods.

2.2.2 ESP

The main focus of [7] is on using incremental evolution to facilitate effective evolution of behaviours to solve complex tasks. However, Gomez and Mikkulainen also introduce a modification to SANE called Enforced Sub-Populations (ESP). They note that when applied to complex tasks, the behaviours that emerge from NE methods are often simple "mechanical" behaviours that lack the necessary complexity to adequately solve the task. Such behaviours are simple to evolve

and show some fitness benefits over random behaviour, so evolutionary searches are easily drawn into parts of the search space that generate such behaviours. For complex tasks, these "mechanical" behaviours are an attractive local optima in the search space that is easy to find, while high quality behaviours are very difficult to find. Thus it is easy for NE methods to converge on these suboptimal behaviours.

In order to overcome this problem, Gomez and Mikkulainen use incremental evoltion: instead of using a single task to evaluate solutions throughout the evolution process, a series of incrementally more difficult tasks is used. Once a sufficient fitness level for a task is achieved or the population converges on a solution for the task, evaluation proceeds on a more difficult version of the task. This is repeated until the population is eventually evaluated on the original goal task. Incremental evolution allows for the evolutionary search to be drawn into parts of the search space which are likely to be able to produce a high quality solution without requiring that a refined solution is found immediately.

Gomez and Mikkulainen based their NE approach on SANE, but modified it by using ESP. While SANE randomly selects neurons from the population to form networks, ESP uses segregated sub-populations for each position in the network. Each position in the network topology has a corresponding sub-population of neurons, and when constructing ANNs a neuron is selected from each sub-population to be used in their corresponding positions. Crossover is also limited to each sub-population: neurons can only be paired with other neurons from their sub-population for crossover. Moriarty in [16] makes the observation that the populations of neurons in SANE converge into species, each of which performs a specialised function. ESP pre-defines these species and also ensures that each species is represented in the final network. Whereas SANE will often selected multiple neurons with the same specialisation or no neurons with a particular specialisation to construct an ANN due to random selection, ESP will always select one neuron from each specialisation. In SANE, evaluations are unstable, since a high-quality neuron could have a low fitness score assigned to it simply because it was always used together with other neurons with the same specialisation. This problem makes it necessary to perform numerous evaluations in order to get accurate fitness values for each neuron. By improving the stability of evaluations, ESP improves the overall performance of the algorithm.

The stability improvement of ESP also makes it possible to evolve recurrant networks, which is difficult using SANE [7]. The performance of recurrant neurons is highly dependant on the neurons to which is it connected. Since SANE uses random selection, recurrant neurons will be connected to neurons with different specialisations in each evaluation. This makes it difficult to accurately determine the fitness of recurrant neurons. Since ESP garuantees that a recurrant neuron will eventually always be connected to neurons with particular specialisations, re-

Method	Pole Balance Attempts				Failures
	Mean	Best	Worst	SD	
GENITOR	1846	272	7052	1396	0
SANE	535	70	1910	329	0
ESP	285	11	1326	277	0

Table 3: Performance of ESP compared to other NE methods on the single-pole balancing problem over 50 simulations

currant neurons can accurately be evaluated and can therefore be reliably evolved.

Another modification that Gomez and Mikkulainen used in [7] is called delta-coding [23]. Delta-coding improves the fine-grained search capabilities of EAs by searching the space around high quality solutions. Once a population has converged, delta-coding is applied by storing the best solution found and evolving a population of Δ-chromosomes. Each Δ-chromosome consists of Δ-values that represent differences to the best solution. The population of Δ-chromosomes is evolved by evaluating the solutions obtained by adding the Δ-values to the best solution. Chromosomes that improve the best solution are selected for crossover, and the evolutionary process proceeds until the population converges. The solution obtained by adding the best Δ-chromosome to the solution becomes the new best solution. Delta-coding can be applied multiple times, with each iteration using the best solution obtained from the previous iteration as the starting point.

Gomez and Mikkulainen applied ESP with delta-coding to a predator-prey simulation in which the predator has limited sensory range. They compared direct evolution with incremental evolution using delta-coding. An initial population is first evolved on a basic task, and then after each task iteration the best solution from that iteration is used as the starting point for delta-coding with evaluations using the next task iteration. They showed that the incremental approach was far more effective than direct evolution in evolving a solution to the most difficult version of the prey-capture task. While direct evolution never managed to produce an adequate solution, the incremental approach was able to find a solution by using 8 incrementally harder versions of the task.

Gomez and Mikkulainen also used ESP (without delta-coding) to evolve a controller for the single-pole balancing problem in order to provide a benchmark of the method. They compared their results to the results given by Moriarty and Mikkulainen in [15]. These results are shown in table 3. Note that they reported the number of attempts required to find a solution and not the time taken to find a solution. Other research has confirmed that ESP is more powerful than many other NE methods [26].

2.2.3 Multi-Agent ESP

In [26], Yong and Mikkulainen introduce Multi-Agent ESP, a modification to ESP in which controllers are evolved for a team of cooperating agents instead of for a single agent. Their research focuses on two questions. Firstly, is it better to use a single neural network as a centralised controller that handles the behaviour of all of the members of the team, or is it better for each team member to be controlled by an autonomous controller? Secondly, is it better for there to be communication between team members or not? These questions were examined using a simple predator-prey system in which the predators must cooperate in order to catch the prey.

In the system used in [26], there are three predators and one prey. The prey is controlled by a simple algorithm while the predators are controlled by one or more evolved ANNs. The predators and the prey move at the same speed, and the prey always moves directly away from the closest predator. The agents can move in one of four directions: N, S, E or W. The environment is a 100x100 toroidal world without any obstacles. The predators always start in the same position, while the prey starts in a random position. The predators have 150 moves in which to catch the prey; if they fail to do so then the trail is deemed unsuccessful. The environment was set up in such a way that it is impossible to capture the prey simply by chasing it, since the prey moves as fast as the predators and there are no walls to trap the prey against. This ensures that the predators must develop a cooperative strategy in order to be successful.

In the certralised controller approach, there is only one network that needs to be evolved, so this was done using the usual ESP method. In the autonomous controller approach, there are three networks that must be simultaneously evolved. During each generation of the evolution process, these networks are formed using ESP normally. They are then evaluated together in the environment and the team is assigned a fitness score. This score is then added to each of the participating neurons' running totals. The following fitness function was used:

$$f = \begin{cases} \frac{d_0 - d_e}{10} & \text{if prey is not caught} \\ \frac{200 - d_e}{10} & \text{if prey is caught} \end{cases}$$

where d_0 is the average initial distance from the prey and d_e is the average final distance from the prey. This function was designed in order to satisfy a few criteria:

1. If the prey is caught then the initial relative positions should not be considered, but the final relative positions should be

2. If the prey is not caught then the distance covered should be considered

Controller	Generations to solve 7/9 benchmark cases		Teams that solved 9/9 benchmark cases
	Mean	SD	
Central	231	116	0/5 Simulations
Distributed	87	22	5/5 Simulations

Table 4: Comparison of central controller to autonomous controllers

3. The function should not use the time taken to catch the prey, since it should be roughly the same in all cases due to the necessity of one of the predators having to travel the length of the board in order to sandwhich the prey between the predators

Similarly to [7], Yong and Mikkulainen also used iterative evolution and delta-coding in their experiments. However, they used delta-coding somewhat differently: they used it at each task transition, after which normal evolution would resume, and they used it in order to re-initialise the populations if their progress stagnated.

Yong and Mikkulainen ran two experiments: the first experiment compared a centralised controller to multiple autonomous controllers, and the second experiment compared agents that communicated to agents with no communication. In this experiment, communication between agents simply means that the agents are able to 'see' each other, i.e. the positions of the other predators forms part of the input to a predator's neural network. In the cases of both the single controller and the autonomous controllers, 30 subpopulations of neurons were used. For the single controller there were 30 hidden-layer neurons, while each of the autonomous controllers had 10 hidden-layer neurons. Each subpopulation contained 100 neurons. They used 1-point crossover in the top 25% of neurons in each population, with the offspring replacing the bottom 50% of the populations in each generation. 40% of offspring are mutated by adding a Cauchy-distributed random value to a randomly chosen weight.

Yong and Mikkulainen used a series of nine tests to measure the success of a team. They considered a team to be successful if they managed to capture the prey in seven out of the nine tests. They ran five simulations of three techniques for 400 generations: central controller, autonomous controllers with communication, and autonmous controllers without communication. The central controller is compared to the autonomous controllers that used communication. The results were given in terms of the number of generations until the best team achieved adequate success in a series of nine tests, and whether or not a team evolved within the 400 generations that could solve all of the nine test cases. Their results are given in tables 4 and 5.

Communication	Generations to solve 7/9 benchmark cases		Teams that solved 9/9 benchmark cases
	Mean	SD	
With	87	22	5/5 Simulations
Without	18	7	5/5 Simulations

Table 5: Comparison of autonomous controllers with and without communication

These results show that autonomous controllers perform significantly better than a centralised controller, being able to reach a particular level of performance close to three times faster. Furthermore, the centralised controller was unable to produce a team that could solve all nine of the test cases in 400 generations. This suggests that cooperative coevolution using autonomous controllers is more powerful than using a single controller. Interestingly, communication between agents was not necessary to produce high quality solutions, and in fact it significantly slowed down evolutionary progress. This is likely because adding communication between the predators is not necessary in order to generate a successful strategy, but doing so triples the number of inputs which greatly increases the size of the search space. However, Yong and Mikkulainen noticed that teams that did not communicate used a single effective strategy in which each predator had a fixed role, whereas teams that did communicate were more flexible and able to use multiple strategies.

2.3 Metrics

As mentioned in section 2.1, premature convergence is a common problem in NE. However, this problem is not restricted to the domain of NE, and various techniques have been developed to encourage diversity in EAs including crowding [6][19] and fitness sharing [6][19]. These techniques rely on being able to identify similar individuals in a population, which necessitates the use of a metric to measure distances between individuals. This has typically been done by using Hamming distance or Euclidean distance in the genotype or phenotype space [8]. However, this approach is problematic when applied to NE due to the nature of the mapping between ANNs and their behaviours. Small differences in structure between networks can result in large differences in behaviour, and networks that appear dissimilar can produce similar behaviours. Thus in the domain of NE it is sensible to measure similarity between individuals based on their actual behaviours, rather than their genotypes or phenotypes. This avoids the competing conventions problem and compares individuals directly on what is of interest to us: their behaviour. However, this approach raises the question of how to effectively compare two behaviours. In this section we examine various metrics that have been used

in the genotype space and behaviour space, and describe some of the ways these metrics have been used.

2.3.1 Generic Metrics

The Hamming distance of two binary strings is simply the number times that the strings have a different bit in the same position. For example, the Hamming distance of the strings 0000 and 0101 is 2. This can be directly applied to genotypes which use a binary encoding [8][19], but it can also be used to create a behavioural distance [8][17]. In [8], Gomez defined Hamming distance based the differences between two action histories (i.e. sequences of actions performed by an agent), while Mouret and Doncieux [17] based their distance on sequences of input-output sets. For a fixed sequence of n inputs, we define a sequence β_x to be the corresponding sequence of actions performed by agent x. We can then define the Hamming distance between the behaviours of agent x and agent y as follows:

$$d_h(\beta_x, \beta_y) = \sum_{i=0}^{n} \left(1 - \delta\left(\beta_x[i], \beta_y[i]\right)\right)$$

where δ is the Kronecker delta defined as

$$\delta(i,j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Thus $d_h(\beta_x, \beta_y)$ simply counts the number of times that the behaviours of agent x and agent y differ. Note that this only gives us a difference between the behaviours of agents for a single sequence of inputs. If the order of inputs matters, such as inputs to a recurrant ANN, then this metric would need to be applied multiple times with a representative sample of input sequences in order to get an accurate measure of the difference between the behaviours of the agents. Hamming distance can be quickly computed and has been shown to be an effective metric for promoting diversity when applied to behaviours [8][17].

Euclidean distance is a simple and widely used distance that can be applied to the genotype space or phenotype space [8][19]. Euclidean distance measures the distance between points in n-space. For genotypes encoded by a vector of n real numbers, we can consider these vectors to be points in n dimensional space and calculate the distance between them as follows:

$$d_e(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}$$

In [14], Li et al. introduce a metric called Normalised Information Distance (NID) that measures the similarity of two arbitrary objects. NID uses the uncomputable notion of Kolmogorov complexity to define a metric. Kolmogorov complexity applies to binary strings, but any object can be represented as such. The Kolmogorov complexity of a string x, $K(x)$, is the length of the shortest binary program that produces x as output when run on a universal computer, such as a universal Turing machine. This is the theoretical lower bound on the compressed size of x. The Kolmogorov complexity of x given y, written $K(x|y)$, is the length of the shortest binary program that produces x when given y as input. NID is defined thus:

$$\mathrm{NID}(x,y) = \frac{\max(K(x|y), K(y|x))}{\max(K(x), K(y))}$$

Since Kolmogorov complexity is uncomputable, we cannot use NID in practice. We can, however, approximate it by using real compression algorithms in the place of Kolmogorov complexity. This gives us the definition of the Normalised Compression Distance (NCD):

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

where C is a compression algorithm. NCD has been shown to be a powerful measure of similarity in a variety of areas. In [14], Li et al. successfully used NCD to produce a mitochondrial genome phylogeny, and also contructed a language tree of Indo-European languages. In [4], Cilibrasi et al. used NCD to classify information in various areas, including taxonomy of viruses, language trees, literature and music. In [8], Gomez used NCD as a metric to measure distances between action sequences of agents.

2.3.2 Non-Generic Metrics

In [2], Balch introduces a behavioural difference that compares agents actions on every posible input vector. The difference between the agents' actions for each input is weighted by the probability that the agents recieve that input. The behavioural difference between agents x and y can be defined as follows:

$$d(x,y) = \sum_{\forall i} \left(\frac{1}{2}(P_i^x + P_i^y)\theta(a_i^x, a_i^y) \right)$$

where i is an input, P_i^x is the probability that agent x recieves input i, a_i^x is the action taken by agent x given input i, and θ is a positive function that measures the difference between two actions. For example, $\theta(i,j)$ could be $1 - \delta(i,j)$, or the absolute difference between the vectors of output activations if the agents are

controlled by ANNs. The benefit of this metric is that is compares agents over the entire set of possible inputs, resulting in a thorough comparison of the agents. However, this may be too computationally expensive when the number of input combinations is large, such as for ANNs with many input nodes. Additionally, it is typically infeasible to apply this metric to agents for which the order of input is relevant, since this would require comparing agents on every possible sequence of inputs.

In [18], Nitschke et al. use an average weight difference metric for sub-populations of neurons for single-layer ANNs in order to regulate recombination both within and between these sub-populations. This metric is based on the generic all-possible-pairs structure for diversity measures described by Wineberg in [24]. Every neuron in one population is paired with every neuron in the other population, and the difference between the corresponding weights for input and output nodes is averaged. The difference is summed over all pairs to produce a final distance between the populations. This provides a measure of the genetic similarity of the populations.

2.3.3 Comparisons of Metrics

In [8], Gomez compares the effectiveness of five behavioural and genotype metrics when used with crowding to evolve an agent controller. The metrics compared were fitness distance, euclidean distance of genotypes, hamming distance of action sequences, relative entropy of action sequences and NCD applied to action sequences. The task of the agent was the Tartarus problem, in which an agent with limited sensory range must move blocks to the edge of a small grid-based world. In terms of fitness vs. generations for various crowding factors, Gomez showed that the behaviour-based metrics performed better than Euclidean distance and fitness distance. NCD had the highest fitness curve, and Hamming distance had the second best fitness curve. Gomez concludes that NCD "produces more fit, general, complex controllers than the other measures".

However, Mouret and Doncieux [17] note that the computational cost of NCD is much higher than that of Hamming distance, while the performance of Hamming distance is only slightly worse than the performance of NCD. They compared task-specific distances, Hamming distance of behaviours and Euclidean distance of genotypes on a variety of tasks with several diversity mechanisms. They found that using behavioural diversity methods substantially improved performance, whereas genotype diversity methods produced meager performance benefits.

3 Method

In this section, we provide a detailed description of the system that was implemented as a testbed to run the experiments. The system contains a predator-prey simulation and an implementation of the neuro-evolution method Enforced Sub-Populations (ESP), all of which is wrapped in a flexible experiment framework that automates evolutionary runs and stores and aggregates the resulting data. The system was implemented almost entirely in Java, with the only exception being a visualisation tool that was implemented in Python. Many of the design choices were made in order to match the system used by Yong and Mikkulainen in [26] as closely as possible, including the design of the simulation environment and the parameters used in the evolution process.

3.1 Simulation Enivronment

The simulation environment executes a simulation of predators attempting to capture a prey using provided behaviours for the predators and the prey. It consists of an $n \times n$ grid (hereafter referred to as the board) on which pieces move. Although the environment allowed for any board size, in our experiments it was always set to be 100×100. Each piece represents either a predator or a prey. Pieces can move only in the cardinal directions (N, S, E or W), with no diagonal movement allowed. The board is toroidal, which means that when a piece moves off the edge of the board, it appears at the opposite edge. For example, if a piece is on the far left column and it moves left, it appears in the same row but in the far right column. This means that there are no boundaries in the environment, which makes it impossible for the predators to capture the prey by cornering it. The predators must therefore surround the prey in order to catch it. In our simulation, catching a prey requires that a predator occupies the same board position as the prey.

The pieces take turns making moves, starting with the predators. Once a piece has moved it is added to the back of the turn queue. In order to facilitate incremental evolution, we needed a way of lowering the speed of the prey relative to the predators. The speed of the prey is specified by a real number $0 \leq s \leq 1$ which determines the prey's speed as a percentage of the predators' speed. Since movement is discrete (i.e. each move changes the position of a piece by one board square), this was implemented by assigning a movement counter to each prey piece. When a prey is at the front of the turn queue, its movement counter is checked. If the counter is ≥ 1 then the piece makes a move and its counter is decreased by 1. Otherwise, the counter is increased by s and the prey does not make a move. This results in the prey making only s times the number of moves that the predators make, which effectively reduces its speed.

The movement of pieces is controlled by behaviours assigned to each piece.

Behaviour is an abstract class containing a getMove method that takes in the state of the board as input and returns an integer representing the move made by the piece. This allows the behaviours of the predators and the prey to be flexibly defined. The predators use a behaviour that is controlled by an ANN. Even though the position of every piece is passed to the behaviour, the predators only uses their own position and the position of the prey to calculate a relative offset which is then used as the only input of the ANN. This design decision was based on the findings of Yong and Mikkulainen [26] described in section 2.2.3. They found that for this predator-prey system, the increased size of the search space that goes with the additional inputs corresponding to the positions of the other predators outweighs the benefits created by the predators being able to see each other. Various strategies were tried for the behaviour of the prey, but ultimately we found that the simple strategy of running directly away from the closest predator was the most effective overall. This also matches the prey behaviour used by Yong and Mikkulainen.

Once a simulation is complete, either because the prey was caught or the turn limit was reached, the results of the simulation are generated. The SimulationResult class includes the number of turns taken, whether or not the prey was caught and vectors containing the final and initial distances of the predators from the prey. This provides the necessary information for a fitness score to be calculated. A detailed description of the fitness function is given in section 3.3.

3.2 Neural Network Specification

Each neural network consists three layers of nodes, each of which is stored in a vector: input nodes, hidden nodes and output nodes. The number of input nodes and output nodes is fixed at 2 and 5 respectively, with the input nodes corresponding to the x and y offset of the prey while the output nodes correspond to the four cardinal directions and an additional fifth node that allows the predator to remain where it is. The number of hidden nodes is changeable, but in our experiments we always used 10. Every node has a vector containing the inputs it recieves from the environment in the case of input nodes or from the nodes in the previous layer. Input nodes and hidden nodes additionally have a vector containing references to the nodes that are connected to their outputs, which are called child nodes. Each hidden node has two vectors of real values that define the weights of its connections to the input and output layers.

When an ANN is run, each hidden node calculates a weighted sum of its inputs, where each input is scaled by the corresponding connection weight. The transfer function is then applied to this sum, which in our case is a sigmoid function. For each output node, this value is scaled by the weight of the appropriate connection and passed on. The output nodes then sum all of their inputs and apply a sigmoid

function to the result, which then becomes the activation value for that output node. The output node with the highest activation determines which direction the predator moves. The sigmoid function used is defined as follows:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Each hidden node is encoded by a genotype. Genotypes each contain a vector of input connection weights, a vector of output connection weights, a fitness value, and a counter that tracks the number of sources that have contributed to the fitness value. The counter is necessary because the fitness value is calculated as the average of the fitness scores assigned to the genotype. Initially, each genotype starts with weights that are pseudo-randomly distributed using a Cauchy distribution centered on 0 with a scale factor of 1. The mutation operator changes a single weight value by adding a normally distributed pseudo-random value. The crossover operator generates a new genotype from two parent genotypes by randomly choosing one of the parents' weights for each connection.

3.3 ESP Implementation

Our implementation of Multi-Agent ESP is largely similar to the method described by Yong and Mikkulainen in [26], but with additional sharing of genetic material between agents. This takes the form of either migration of individual genotypes between sub-populations of different predators, or the application of an inter-population crossover operator. These are managed by behavioural and genetic metrics that limit the sharing of genetic material to predators that are sufficiently similar genetically and/or behaviourally. The intuition behind this modification is that it allows predators that develop the same strategic role to share their evolutionary progress, meaning that the same high-quality genetic material does not have to be separately evolved multiple times for different predators. We used incremental evolution, but unlike previous research we only used two versions of the prey-capture task: the easier version of the task has a prey that moves at 10% of the speed of the predators, while in the harder version the prey moves at the same speed as the predators. Numerous control parameters can be used to modify the evolution process in various ways, including turning on or off migration or inter-population crossover, controlling how often these operators are used, applying crossover periodically instead of in every generation or even disabling it entirely, adjusting the similarity threshholds for the metrics, and much more.

At the start of each evolutionary run, a population of random genotypes is initialised for each predator. Each population consists of 10 sub-populations (corresponding to the 10 hidden nodes per ANN) which each contain 100 genotypes, making a grand total of 3000 genotypes. At the beginning of each generation,

the environment is set up using a random position for the prey. At this point, 10 sets of trials are run. In each set of trials, each genotype is used once together with nine other genotypes from the other sub-populations to construct an ANN. Thus in every set of trials 100 different ANNs are constructed and evaluated in a simulation, with the resulting fitness score being passed back to the participating genotypes. Once the trials are comleted, crossover and mutation are applied to each sub-population. During crossover, the genotypes in each sub-population are ranked according to fitness, then the genotypes in the bottom 50% are replaced with offspring created by randomly pairing up genotypes in the top 25%. After crossover, every genotype has a 40% chance to be mutated by adjusting a single weight. Elite predators are constructed from each population using the genotypes with the highest fitness in each sub-population. These predators are then run through a series of tests using a representative selection of initial prey positions. If the test score is sufficiently high, the current task is considered to be solved. If the current task is the easier version, then evolution continues on the harder version. If the harder version is solved, then evolution halts.

At this point in each generation, migration or inter-population crossover may be applied, depending on whether the operator is enabled and whether the generation number is divisible by the interval at which the operator is used. One or both of the metrics are applied to control the sharing of genetic material. In the case of the behavioural metric, each pair of predators is checked for similarity, while the genotype metric measures the similarity of sub-populations. If both metrics are enabled, the behavioural metric is applied first and then only predators that are sufficiently similar have their sub-populations compared using the genotype metric. Sharing occurs between the genetically similar sub-populations of the behaviourally similar predators (but no sharing occurs between the sub-populations of a single predator). If the behavioural metric is disabled, then the genotype metric gets applied to every pair of sub-populations from different predators and the sharing operator is applied to the sufficiently similar sub-populations. If the genotype metric is disabled, then genetic material is shared between every sub-population from similar predators. In this case, where only the behavioural metric is used, the sub-populations are paired up at random. When the migration operator is used, two random genotypes from each of the selected sub-populations are sent to the other sub-population, replacing the worst two genotypes in the recieveing sub-population. The inter-population crossover operator pairs up genotypes in the fittest 25% of each sub-population for crossover. The offspring that are produced are added to both sub-populations, replacing the weakest 25% of each sub-population (since each pair produces a single offspring).

We chose to use non-generic metrics in this project. We decided to use the behaviour distance described by Balch in [2]. The reason to use this over another

metric like Hamming distance or NCD is that it provides a thorough comparison of behaviours by comparing the output under all possible inputs. For large input spaces this is infeasible, but our ANNs use only two inputs, each of which has a range of only 100 possibilities. This makes the compuational cost of using this behaviour distance negligible, which is its main drawback. Using this metric allows us to compare the behaviours of agents completely without having to construct representative action sequences for the agents. For the genotype metric, we used the average weight distance used by Nitschke et al. in [18]. This seemed like the obvious choice given that our genotype was not encoded as a string but rather as a vector of weights. While we could have used the average Euclidean distance between every pair of neurons to measure distance between sub-populations, the value that is produces does not have an intuitive interpretation, whereas the value produced by the average weight distance can be more easily understood. Therefore we chose to use the average weight distance rather than Euclidean distance.

3.4 Experiment Framework

The Experiment Framework class allows for easy automation of evolutionary runs. Each experiment consists of a number of repetitions of the evolution process, and each experiment can use entirely different parameters. The results from each repetition are stored and averaged to provide a summary for each experiment. The statistic that is of primary interest is the average number of generations required until convergence.

The framework first reads a simple script file that specifies the number of experiments to run, the repetitions of each experiment and the parameters for each experiment. The experiments are then executed in the order they were specified. Finally, the results from each repetition and a summary for each experiment are written to a text file.

References

[1] Charles William Anderson. *Strategy learning with multilayer connectionist representations*. GTE Laboratories, 1987.

[2] Tucker Balch. Social entropy: a new metric for learning multi-robot teams. 1997.

[3] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *Systems, Man and Cybernetics, IEEE Transactions on*, (5):834–846, 1983.

[4] Rudi Cilibrasi and Paul M B Vitányi. Clustering by compression. *Information Theory, IEEE Transactions on*, 51(4):1523–1545, 2005.

[5] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, January 2008.

[6] David E Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 41–49. L. Erlbaum Associates Inc., 1987.

[7] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317–342, 1997.

[8] Faustino J Gomez. Sustaining diversity using behavioral information distance. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 113–120. ACM, 2009.

[9] Frederic Gruau. Genetic synthesis of boolean neural networks with a cell rewriting developmental process. In *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*, pages 55–74. IEEE, 1992.

[10] Frédéric Gruau. Automatic definition of modular neural networks. *Adaptive behavior*, 3(2):151–183, 1994.

[11] Frederic Gruau, Darrell Whitley, and Larry Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 81–89. MIT Press, 1996.

[12] Christian Igel. Neuroevolution for reinforcement learning using evolution strategies. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 4, pages 2588–2595. IEEE, 2003.

[13] Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems Journal*, 4:461–476, 1990.

[14] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul M B Vitányi. The similarity metric. *Information Theory, IEEE Transactions on*, 50(12):3250–3264, 2004.

[15] David E Moriarty and Risto Mikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine learning*, 22(1-3):11–32, 1996.

[16] David Eric Moriarty. Symbiotic evolution of neural networks in sequential decision tasks, 1997.

[17] J-B Mouret and S Doncieux. Encouraging behavioral diversity in evolutionary robotics: an empirical study. *Evolutionary computation*, 20(1):91–133, January 2012.

[18] Geoff S Nitschke, Martijn C Schut, and A E Eiben. Collective neuro-evolution for evolving specialized sensor resolutions in a multi-rover task. *Evolutionary Intelligence*, 3(1):13–29, 2010.

[19] B. Sareni and L. Krahenbuhl. Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation*, 2(3):97–106, 1998.

[20] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, January 2002.

[21] Christopher J C H Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[22] Darrell Whitley, Stephen Dominic, Rajarshi Das, and Charles W Anderson. *Genetic reinforcement learning for neurocontrol problems*. Springer, 1994.

[23] Darrell Whitley, Keith Mathias, and Patrick Fitzhorn. Delta coding: An iterative search strategy for genetic algorithms. In *ICGA*, volume 91, pages 77–84, 1991.

[24] Mark Wineberg and Franz Oppacher. The underlying similarity of diversity measures used in evolutionary computation. In *Genetic and Evolutionary Computation—GECCO 2003*, pages 1493–1504. Springer, 2003.

[25] Xin Yao. Evolving artificial neural networks. In *Proceedings of the IEEE*, volume 87, pages 1423–1447. IEEE, 1999.

[26] Chern Han Yong and Risto Miikkulainen. Cooperative coevolution of multi-agent systems. *University of Texas at Austin, Austin, TX*, 2001.