Meta-NEAT

**ABSTRACT**

Though neuroevolution has been among us for quite some time now, most recent focus in the field of neural networks has almost exclusively been on classification problems and not on behaviour learning. NEAT (1) and subsequent implementations have been proven to be well suited for a wide variety of tasks but little emphasis has been put on classification problems through neuroevolution. Because of this shift towards backpropagating convolutional and deep neural networks, neuroevolution was set aside and only scarcely used in computer games and evolutionary robotics. This paper introduces a new implementation to NEAT through the use of current, up-to-date technologies and a novel crossover technique which implies incest. On top of this, a meta-evolution algorithm is employed over the initial evolutionary algorithm in order to analyse and find hidden structure underneath all the different tuning parameters of the initial GA. Thus, the *meta*-GA basically evolves configurations containing parameters representing the mutation probabilities, population size and other such constants of a classical GA. The role of such a meta-analysis is to find configurations which minimize the number of generations needed for the evolution to converge to a desired solution and capture the intricacies of those configurations with regards to particular problems.

**CCS Concepts**

• **Emerging technologies ➝** **Biology-related information processing ➝ Neural Systems**

• **Machine Learning ➝** **Learning paradigms ➝ Reinforcement learning**

• **Machine learning approaches ➝ Neural Networks**

**Keywords**

Neuroevolution, network topologies, genetic algorithms, neural networks, classification, behaviour.

# INTRODUCTION

Since the mid to late 1990s in papers like (2), (3) and (4), promising results have been obtained in the field of neuroevolution by solving delicate reinforcement learning tasks. The core concept of neuroevolution is having a pool of neural networks as the population of a genetic algorithm and searching through that pool of behaviours for a network that performs well for a given task. A rather promising approach as shown in (5), neuroevolution proved to be faster and more efficient than classical reinforcement learning methods like Adaptive Heuristic Critic and Q-Learning on problems like single point balancing and robot arm control.

Traditionally, neuroevolution techniques started out with a fixed topology and evolved only the weights of the networks which contained fully connected neurons. Fixed topology NE techniques have the grave disadvantage of considering only the weights as being important to the behaviour of a network, however, they are far from being the only contributing factor to a network’s behaviour. The structure or topology of a network also deeply affects and influences its behaviour. (6) introduces for the first time structurally unconstraint recurrent connections in its evolutionary approach which, in contrast to constructive and destructive algorithms employs a population of networks and uses a fitness function’s unsupervised feedback to guide search through network space. Due to this enhancement, NE became a natural choice for learning non-Markovian tasks (7). The goal of (1) was to demonstrate that if done right, evolving structure along with connection weights can significantly enhance the performance of NE.

Evolving both weights and structure comes with several technical difficulties and challenges, the first being representing and encoding the network structure and using that encoding as an individual in an evolutionary algorithm. The solution to this encoding problem has to take into account the possibility of two disparate topologies doing cross over, and hence has to be able to realize such a cross over in a meaningful but still elegant way. Another challenge would be protecting innovation which can initially hinder the fitness function of an individual as thus cause it to be prematurely removed from the population before being able to optimize its new structure. Starting with a random structure might impose the fitness function to take into account the complexity of the network’s topology and try to minimise that, however, then the question of how much of a penalty should the fitness function apply on a large structure arises. The question of whether or not the initial topology of the first individuals should be random or not is also an interesting problem as it dictates the presence or absence of a reduction mutation which should also be implemented alongside the neuron/synapse addition counterparts. (1) provides a good answer to all those questions.

In (8) meta-genetic algorithms are used to tune parameters of GA in order to find lowest energy molecular conformers. Similarly, the purpose of the meta-analysis done by the genetic algorithm running on top of the network topology evolutionary algorithm is to discover parameter configurations which, if used for specific problems, will render a fast convergence rate to a desired solution. Other aspects such as minimal structure or synapse density or sparsity might also be of interest, however, some of those aspects are covered by the NEAT algorithm itself. (E.g. starting off from minimal topologies implies a more controlled growth and can almost guarantee that the algorithm converges towards minimal structure networks)

# Background

Many approaches to neuroevolution have been developed throughout the years. (9), (10) and (11) are only a few examples of such algorithms which describe how *Topology and Weight Evolving Artificial Neural Networks* (TWEANNs) should be implemented. Much progress on the matter has been done since and in this section we will analyse a few novel approaches to NE and where it has been most helpful.

## NEAT

Neuro Evolution of Augmenting Topologies is the currently the reference paper and base approach of nearly all neuroevolution techniques. Fully described in (1), NEAT gives answers and solutions to the above mentioned questions related to both encoding and protecting innovation and to the initial minimal structure hypothesis which states that it is much more efficient and elegant to start off with a population of networks which are all of minimum topological complexity.

To answer the encoding problem, the basic NEAT approach choses a direct encoding technique. In *direct* encoding schemes, the genotype directly maps to the phenotype. That is, every neuron and connection in the neural network is specified directly and explicitly in the genotype. In contrast, in *indirect* encoding schemes, the genotype specifies indirectly how the network should be generated. Indirect encodings are often used to achieve several aims (12) (13):

* Allow recurring structures or features in the network to form
* Compression of phenotype to a smaller genotype, providing a smaller search space
* Mapping the search space (genome) to the problem domain

Indirect encoding is used in future NEAT augmentations which will be described shortly.

Innovation in NEAT appears through structural mutation (1). The two types of mutators responsible for such mutations are the *Add-Node* mutator and the *Add-Synapse* mutator. At first, whenever either a neuron is added between two already existent ones or a synapse is created between two nodes which previously had no linking synapse, the overall fitness of the individual might drop as the synapse weights are not optimized. This will lead to a premature elimination of that individual from the population pool. In order to give such individuals a chance to optimize their new structure, NEAT proposes speciation, a technique in which individuals compete between each other in their own species. An individual belongs in a species if it is topologically similar to the representative of that species (a random individual chosen from that species). Although it may initially seem like a topology matching problem, it is not. By using direct encoding, each gene which corresponds to a synapse is associated an *innovation* number. Two genes with the same historical origin must represent the same structure (although with possibly different weights). For example, when a synapse between nodes ni and nj is added as a gene, a global innovation number is either incremented in case that synapse between those two specific neurons never occurred in any individual, or is fetched from an *innovation number map* which holds the associated innovation number for a specific synapse residing between two neurons. This way, the topological distance between two individual is transformed from a complex topological mapping problem to a simple innovation number distance calculation. As a by-product, the *competing conventions* problem is also resolved using this technique.

NEAT starts off with minimal topologies as the solution itself to tend towards minimality. If the population begins with no hidden nodes and grows structure only as it benefits the solution (all this whilst being protected through speciation), there is no need for ad-hoc fitness modification which has the purpose of minimizing the network size. Another advantage of starting minimally is that the system searches for the solution in the lowest-dimensional weight space possible over the course of all generations. Thus, the goal becomes not to minimize only the final product, but all intermediate networks along the way as well (1). All this results in dramatic performance gains.

## HyperNEAT

As an augmentation of NEAT, HyperNEAT (14) tries to narrow the gap between natural evolution discovered in intelligent brains which can contain billions of neurons with trillions of connections and neuroevolution. HyperNEAT employs an indirect encoding called connective Compositional Pattern Producing Networks (connective CPPNs) that can produce connectivity patterns with symmetries and repeating motifs by interpreting spatial patterns generated within a hypercube as connectivity patterns in a lower-dimensional space. The advantage of this approach is that it can exploit the geometry of the task by mapping its regularities onto the topology of the network, thereby shifting problem difficulty away from dimensionality to underlying problem structure. Furthermore, connective CPPNs can represent the same connectivity pattern at any resolution, allowing ANNs to scale to new numbers of inputs and outputs without further evolution.

Compositional pattern-producing networks are a variation of ANNs that differ in their set of activation functions and how they are applied. While ANNs often contain only sigmoid functions and sometimes Gaussian functions, CPPNs can include both types of functions and many others. The choice of functions for the canonical set can be biased toward specific types of patterns and regularities. For example, periodic functions such as sine produce segmented patterns with repetitions, while symmetric functions such as Gaussian produce symmetric patterns. Unlike typical ANNs, CPPNs are applied across the entire space of possible inputs so that they can represent a complete image. Since they are compositions of functions, CPPNs in effect encode images at infinite resolution and can be sampled for a particular display at whatever resolution is optimal.

## Usages of NEAT

The most accurate measurement to date of the mass of the top-quark was computed by a large team at the Tevatron collider using NEAT. In (15), events are selected with a neural network which has been directly optimized for statistical precision in top quark mass using neuroevolution. Here, the neuroevolution technique is preferred as it combines the parametrization of an arbitrary multivariate selection described by a neural network with an evolutionary minimization approach to search for the network weights and topology which in turn optimizes an arbitrary metric. Neuroevolution is used in order to optimize for minimal expected statistical uncertainty in the top quark mass measurement.

In a maybe more unusual setup, (16) presents a general approach for simulation and controlling a human n character that is riding a bicycle. They simulate the bicycle and the rider as an articulated rigid body system. The rider is controlled by a policy that is optimized through offline learning. They also apply policy search to learn the optimal policies, which are parameterized with splines or neural networks for different bicycle maneuvers. Neuroevolution of Augmenting Topology (NEAT) is used to optimize both the parametrization and the parameters of our policies.

In (17) we find an approach of procedurally generating constantly renewing game content with the purpose of having players remain engaged more. Two new novel technologies are introduced that take steps toward achieving this ambition: 1) a new algorithm called *content-generating Neuro Evolution of Augmenting Topologies* (cgNEAT) which automatically generates graphical and game content while the game is played, based on the past preferences of the players and 2) *Galactic Arms Race* (GAR), a multiplayer video game, constructed to demonstrate automatic content generation in a real online gaming platform. In GAR, which is available to the public and playable online, players pilot space ships and fight enemies to acquire unique particle system weapons that are automatically evolved by the cgNEAT algorithm. A study of the behavior and results from over 1,000 registered online players shows that cgNEAT indeed enables players to discover a wide variety of appealing content that is not only novel, but also based on and extended from previous content that they preferred in the past. Thus GAR is the first demonstration of evolutionary content generation in an online multiplayer game.

(18) applies an indirect encoding to the problem of scalable Go that can evolve a solution to 5x5 Go and then extrapolate that solution to 7x7 Go and continue evolution. The scalable method is demonstrated to learn faster and ultimately discover better strategies than the same method trained on 7x7 Go directly from the start. This version of a Go AI player precedes the now famous AlphaGo player from Google’s DeepMind detailed in (19) which eventually a Go grand-master in a series of games. The two approaches however are different and their difference is not a topic for discussion here.

As to even further show the width of applicability of NEAT or related approaches, (20) presents a new musical representation that contains almost no built-in knowledge but always even musically untrained users to generate polyphonic textures that are derived from the users’ own initial compositions. This representation, called functional scaffolding for musical composition (FSMC), exploits a simple yet powerful property of multipart compositions: The pattern of notes and rhythms in different instrumental parts of the same song are functionally related. That is, in principle, one part can be expressed as a function of another. Music in FSMC is represented accordingly as a functional relationship between an existing human composition, or scaffold, and a generated set of one or more additional musical voices. To understand the idea behind FSMC, consider the proposition that if different simultaneous instrumental parts in the same composition were not somehow related to each other, they would probably sound inappropriate together. This relationship can be conceived as a function that describes how one part might be transformed into another. That is, theoretically there exists a function that can transform one sequence of notes and rhythmic information into another. The idea in FSMC is to exploit this fact by literally evolving the function that relates one part to another. That way, instead of searching for a sequence of notes, FSMC can search for a transforming function that bootstraps off the existing parts (i.e. called the scaffold) to generate the additional voices. In effect, FSMC is the hidden function that relates different simultaneous parts of a composition to each other.

The problem of *evolvability* (evolution’s capability of finding innovative phenotypes) is addressed in (21) where the author states that numerous explanations for the origins of evolvability have been proposed, often differing in the role that they attribute to adaptive processes. He then proceeds to provide a new perspective on these explanations through experiments which simulate evolution in gene regulatory networks, revealing that the type of evolvability in question significantly impacts the dynamics that follow. The results reconcile a large body of work across biology and inform attempts to reproduce evolvability in artificial settings. In particular, while adaptive processes result in evolvable individuals, processes that are either neutral or that explicitly encourage divergence result in evolvable populations. Furthermore, evolvability at the population level proves the most critical factor in the production of evolutionary innovations, suggesting that nonadaptive mechanisms are the most promising avenue for investigating and understanding evolvability.

A simple evolutionary optimization which can rival stochastic gradient descent in neural networks is presented in (22). Here, the author catches the nuance that EA have been rather absent in deep learning and says that one explanation for this is that neural networks have become so high-dimentional that evolution with its inexact gradient cannot meci the exact gradient calculations of backpropagation. Furthermore, the author states that the evaluation of a single individual in evolution on the big data sets now prevalent in deep learning would present a prohibitive obstacle towards efficient optimization. In (22) the author challenges those views suggesting that Eas can be made to run significantly faster than previously thought by evaluating individuals only on a small number of traning examples per generation. Surprisingly, using this approach with only a simple EA (called the limited evaluation EA or LEEA) is competitive with the performance of the state-of-the-art SGD variant RMSProp on several benchmarks with neural networks with over 1,000 weights.

# Approach

On top of a brand new implementation of NEAT we will discuss adding a meta-analysis approach to NEAT as a whole. This new evolutionary algorithm will situate itself one level above our neuroevolving GA and its search space will consist of different configurations on which NEAT will run on. Afterwards, we will analyse the effect of *incest*, a novel crossover idea which has the purpose of creating two offspring instead of just one at any given NEAT crossover procedure which, normally produces just one offspring.

## Meta-analysis of NEAT

One of the major differences which set genetic algorithms evolving neural network topologies aside from the rest of GAs is their complexity when it comes to the number of different *mutations* which can be applied on any individual from the population. Because the encoding represents something much deeper in meaning than simple bit strings or number strings, NEAT implementations can offer a very wide range of *mutators/alterers* which can all have different parameter settings. All these settings influence the convergence rate of NEAT and heavily dictate how fast or slow (generation wise) the best individual in the population is found. Therefore, an analysis into these configurations can give great insight on how to best optimise the search of a topology by actually transposing the whole problem one level higher.

### Meta-fitness

Because this meta-analysis will actually involve a different GA applied on top of NEAT, a fitness function is required. The desired effect of such an analysis is to find parameter configurations which render, generation wise, a faster convergence rate of individuals. Even though, as mentioned in (1), we might be interested in punishing larger topologies based on their number of neurons or synapses, we are still delegating that responsibility directly to NEAT and not imposing such a punishing restriction in this fitness function either. The meta-fitness function will only be interested in the generation at which an individual was found and will try to minimise this or maximise its negative counterpart. Therefore, the simplest form of such a fitness function can be summarised like in Equation 1:

Equation

In this context, a phenotype is actually a neural topology which reached the desired fitness in our NEAT implementation and hence is a result of one of the configurations.

### Meta-dimensions

In our implementation of NEAT, there are 5 mutation operators and one crossover operator:

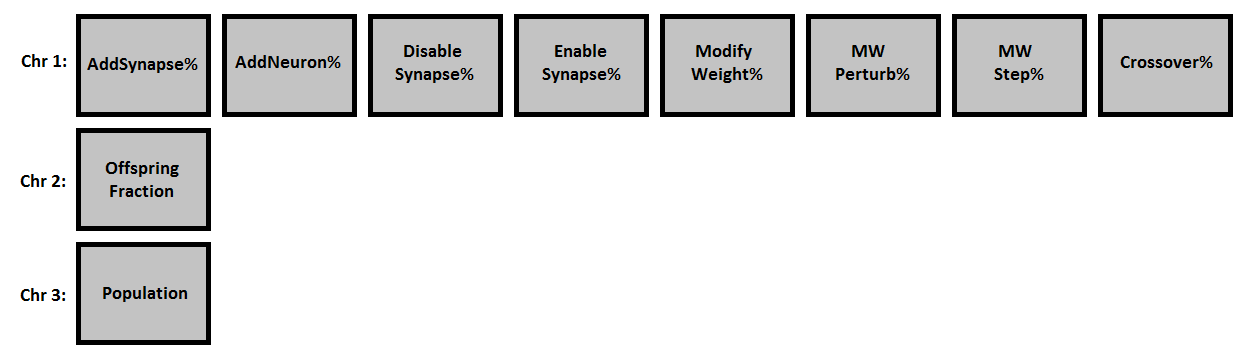
* Add Synapse mutator
* Add Neuron mutator
* Disable Synapse mutator
* Enable Synapse mutator
* Modify Weight mutator
* NEAT Crossover

With the exception of the *Modify Weight* mutator, all others have a simple probability parameter which theoretically ranges from 0 to 1. In addition to this probability parameter, the *Modify Weight* mutator has two more parameters. Assuming there is a *Pw%* chance that a weight modification mutation occurs, in case it happens, each weight has a PW-perturb% chance of being uniformly perturbed with just PW-step% of its weight and a (1 – PW-perturb)% chance of being assigned a new random value. As all these probabilities are between 0 and 1, they can be encoded in a separate chromosome containing 8 different floating-point numbers restricted in the (0, 1) interval.

Another dimension which can be taken into consideration is the offspring fraction. The offspring fraction represents the percentage of individuals which pass on to the next generation as a result of mutations or alterations rather than being directly transferred as survivors from the previous generation. The offspring fraction was chosen to range between 0.1 and 0.99 hence it needed to occupy a separate chromosome in our meta-individual as to not crossover with other chromosomes which contain different genes.

Lastly, the population size was chosen as a third dimension and itself resides in yet another different chromosome. Its lower bound is 20 and upper bound chosen to be 250. This population size refers not to the population size of the meta-GA but rather the population size of the NEAT implementation.

In Figure 1 we more clearly see the asymmetric genotype structure of the meta-genotype.



Figure

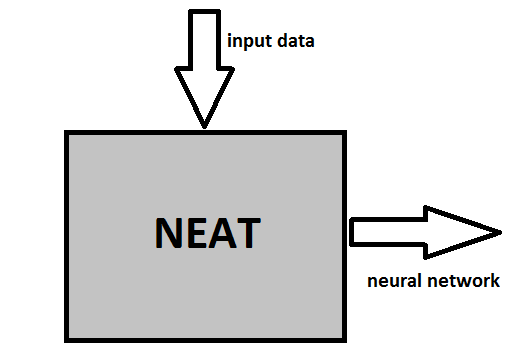
### Meta-evolution parameters

As with any GA, the meta-analysis algorithm placed on top of NEAT needed some evolution parameters of its own. As opposed to the NEAT implementation which had a survivor selector which applied elitism, both the survivor selector and the offspring selector of the meta-evolution GA were chosen to be a *tournament selector*.

As mutators, a *value mutator* was chosen alongside a *single point crossover*. It is worth mentioning that the crossover takes only genes from the same chromosome from two different individual, hence there is no risk of crossing-over the *Offspring Fraction* with the *Enable Synapse %.*

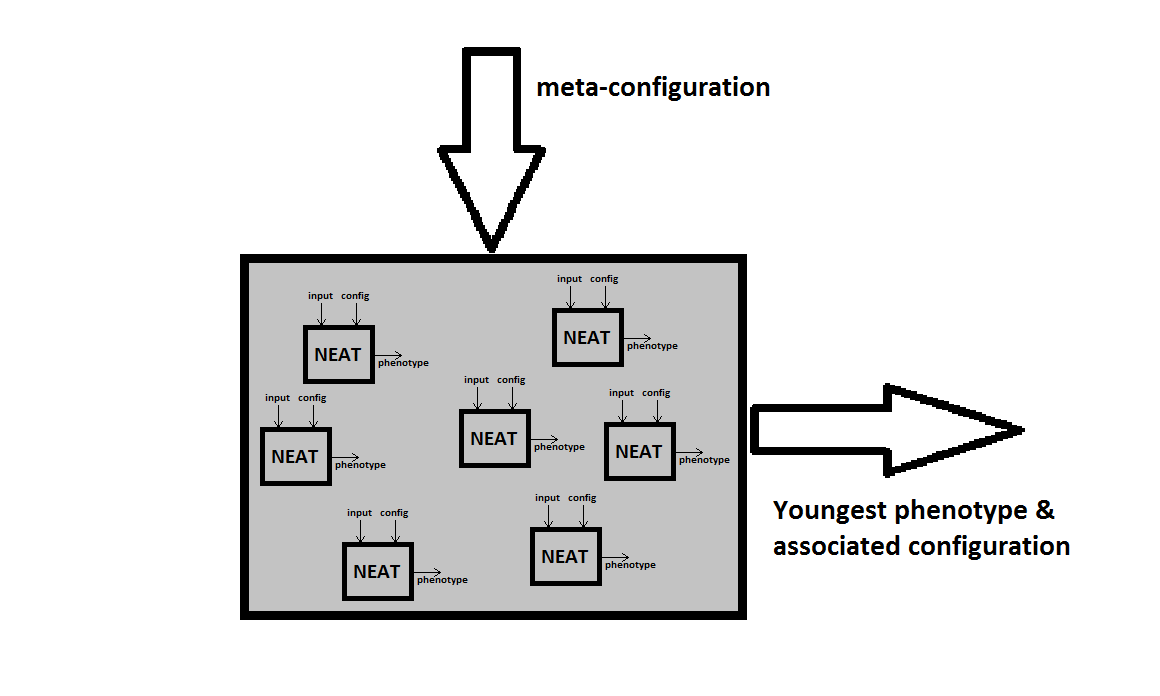
### Meta-flow

Let us consider a black box view of NEAT as shown in Figure 2.



Figure

There is some input data fed to NEAT which contains the fitness function for that data, the evolution parameters etc. The best phenotype in terms of fitness is chosen and outputted from NEAT, decoded as a neural network. As of yet, there is no way to automatically extract *best tuning* parameters for the evolution. Surely, one can simply tune those parameters manually and through trial and error simply observe the outcome by examining the phenotype’s age; however that proves to be a very cumbersome task. Furthermore, there are at least 8 parameters which all need tuning in order to more quickly produce a suitable individual therefore being countless combinations which might prove to be good for this specific task. This is where our meta-GA comes into play. First, we extract the configuration which NEAT runs on such that it can be fed from the outside and not be bound to the algorithms itself. Then, we construct a genetic algorithm which has as individuals different such configurations which are fed to instances of NEAT. In Figure 3 we can see a diagram of this which has many NEAT instances inside a now larger meta-GA.



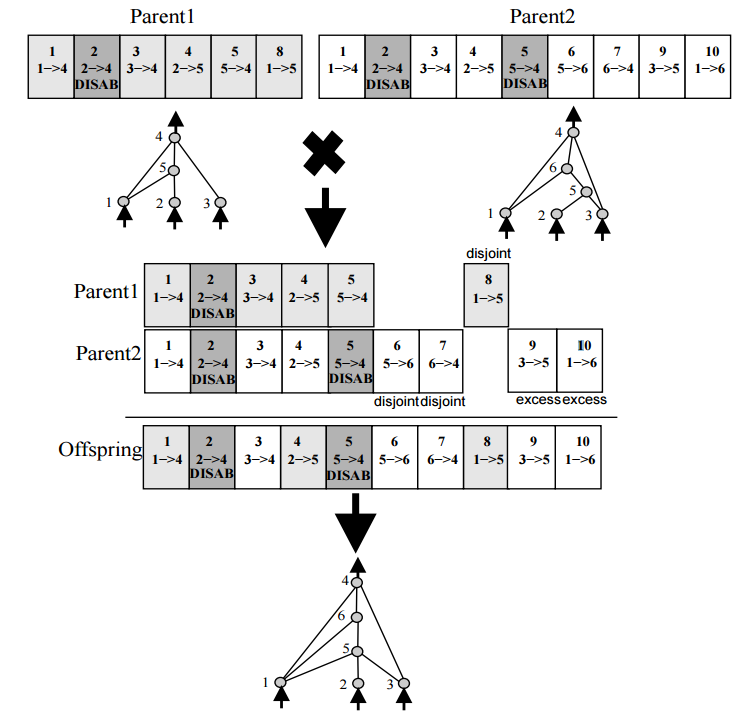
Figure

The meta-configuration fed to this GA contains the bounds of the NEAT parameters discussed in section 3.1.2 and the meta-evolution parameters discussed in section 3.1.3.

We will discuss the results of the meta-analysis in chapter 4.

## Incestuous NEAT crossover

In Figure 4 from (1) we see the proposed method of crossover in NEAT. Two parents create *just one* offspring by combining their genes. In almost any other classical crossover implementations in any genetic algorithms, crossover produces two offspring. *NEAT on the other side normally produces just one offspring during the crossover of two individuals.* So how can NEAT be changed to produce not one, but two offspring. Figure 4 shows the initial crossover technique proposed in (1).



Figure

We will now propose a slightly different technique which will result in two offspring instead of one. The new technique will involve *incest* between the initial offspring and both its parents.

After the two parents are selected for crossover and their offspring is created as shown in Figure 4, two additional crossover operations are done. One between the *offspring and Parent 1* and the other between the *offspring and Parent 2.* We are now left with 3 offspring, two of which are the result of incest between the initial offspring and its parents.

In order to chose only two of those three offspring, we evaluate and sort them by their fitness function, taking the first two most fit individuals and placing them as offspring in our population. Even though some work and effort has been invested into proving that, in most cases when it comes to GA incest leads to consistently worse results, we are curious to see whether or not NEAT’s speciation technique addresses this issue and manages to solve it. The results of this new approach will be analysed in the next chapter.

# Analysis

In this chapter we will go through the environments, parameters and test data used in our analysis. We will first present the three different problems on which the algorithm was tested, afterwards going through the settings used for each problem. Thirdly, we will analyse the results and lastly briefly mention the technologies used in this new implementation of NEAT.

## Environments

We chose 3 different environments/problems to test our algorithm on.

The first obvious choice was the simple problem of modelling a XOR gate. The advent of multilayer neural networks sprang from the need to implement the XOR logic gate. Early perceptron researchers ran into a problem with XOR. The same problem as with electronic XOR circuits: multiple components were needed to achieve the XOR logic. With electronics, 2 NOT gates, 2 AND gates and an OR gate are usually used. With neural networks, it seemed multiple perceptrons were needed. To be more precise, abstract perceptron activities needed to be linked together in specific sequences and altered to function as a single unit. Thus were born multi-layer networks. It was only natural to choose XOR as a starting problem as it easily tests NEAT’s capability of evolving networks from minimal structure as XOR requires hidden nodes in ordered to be modeled correctly by a neural network.

The second slightly more complex scenario was that of a classification problem, albeit a simple one. The Iris Data Set[[1]](#footnote-1) was chosen to test NEAT on a classification problem as well. The fitness function for this problem used the tenfold cross validation technique to grade its phenotypes.

The last problem was that of learning to play Tic-Tac-Toe. An individual would play against a *perfect* computer player for 40 rounds. It would have its fitness incremented by 2 in case it won and by 1 in case the match concluded in a draw. Of course, as the predefined computed player was perfect, there was no way in which the network could possibly end up winning, hence the fitness function threshold was set to 40. If a phenotype managed to achieve a fitness of 40, that would mean that it got 40/40 draws whilst playing against a perfect player, hence making it *perfect* as well.

## Settings

Similar settings were used for testing both the meta-evolutionary algorithm and the incestuous crossover technique. The main analysis goal was to peek at the convergence rate of the individuals, therefore, only be either running NEAT on the same fixed configuration many times and averaging the convergence rate or by using the meta-GA could such a result be extracted.

### Standard NEAT configuration

The standard NEAT configuration is used whenever the meta-algorithm is not. It contains common sense parameter settings for the NEAT algorithm. Table 1 shows the settings were used for the *standard NEAT configuration*:

Table – standard NEAT configuration

|  |  |
| --- | --- |
| Configuration | Value |
| Population size | 150 |
| Offspring fraction | 85 % |
| Add Synapse Mutation | 2 % |
| Add Neuron Mutation | 2 % |
| Disable Synapse Mutation | 2 % |
| Enable Synapse Mutation | 4 % |
| Modify Weight Mutation | 80 % |
| Modify Weight Perturb | 90 % |
| Modify Weight Perturb-step | 10 % |
| Crossover | 20 % |

Note that some NEAT-specific configurations like the *speciation threshold* or *maximum staleness* are not present in this table as they are not in the meta-GA configuration genotype. However, for completion, their value ranged from 1.4 – 2.5 in case of the speciation threshold and from 25 – 70 for the maximum staleness. Also, the three species-compatibility weights used to compute the distance between two NEAT individuals were fixed to:

* Excess genes: 1.0
* Disjoint genes: 1.0
* Weight diff: 0.8

### Meta-NEAT configuration

Our meta-algorithm also needed a few configuration settings of its own. As mentioned in 3.1.3, the survivor selector and the offspring selector of the meta-evolution GA were chosen to be a *tournament selector*. The tournament selector was set to hold tournaments of fixed size 3. The *value mutator* had a probability of 1.5% and the *single-point crossover* a probability of 20%.

Table 2 shows the bounds of the configurations imposed upon the actual NEAT configuration but present in the *meta-NEAT* configurations.

Table – meta-NEAT bounds

|  |  |  |
| --- | --- | --- |
|  | Lower bound | Upper bound |
| Mutation probability | 0.01 % | 100 % |
| Offspring fraction | 1 % | 99 % |
| Population size | 20 | 250 |

Basically, we imposed bounds on the genes’ value from all the three different chromosomes present in each genotype from *meta-NEAT.* Besides these three settings, we have also tested *meta-NEAT* on a fixed population size of 20 and for a maximum of 10 generations.

There were no particular extra-settings needed for the incestuous crossover tests, hence we have used the same settings as from 4.2.1 and 4.2.2.

## Results

We will first take a look at the meta-NEAT results. We analyze meta-NEAT by trying to find the configuration which most quickly converged to a solution from the multitude of configurations tried out. All the meta-NEAT analysis is done *without* the incestuous crossover technique. We will only afterwards look at the results for the new crossover technique. The three environments (XOR, Iris classification, Tic-Tac-Toe) are in both cases analyzed separately.

### Meta-NEAT results

We will now compare the *Standard NEAT configuration* presented in 4.2.1 with the obtained average values from all configurations which most quickly led to convergence after the Meta-NEAT was run on all 3 problems.

Table – XOR

|  |  |  |
| --- | --- | --- |
| Configuration | Std. Value | Meta-NEAT |
| Population size | 150 | 185 |
| Offspring fraction | 85 % | 82 % |
| Add Synapse Mutation | 2 % | 61 % |
| Add Neuron Mutation | 2 % | 53 % |
| Disable Synapse Mutation | 2 % | 49 % |
| Enable Synapse Mutation | 4 % | 55 % |
| Modify Weight Mutation | 80 % | 30 % |
| Modify Weight Perturb | 90 % | 50 % |
| Modify Weight Perturb-step | 10 % | 50 % |
| Crossover | 20 % | 60 % |

Table – Iris Classification

|  |  |  |
| --- | --- | --- |
| Configuration | Std. Value | Meta-NEAT |
| Population size | 150 | 180 |
| Offspring fraction | 85 % | 61 % |
| Add Synapse Mutation | 2 % | 55 % |
| Add Neuron Mutation | 2 % | 59 % |
| Disable Synapse Mutation | 2 % | 65 % |
| Enable Synapse Mutation | 4 % | 45 % |
| Modify Weight Mutation | 80 % | 50 % |
| Modify Weight Perturb | 90 % | 62 % |
| Modify Weight Perturb-step | 10 % | 42 % |
| Crossover | 20 % | 41 % |

Table – Tic-Tac-Toe

|  |  |  |
| --- | --- | --- |
| Configuration | Std. Value | Meta-NEAT |
| Population size | 150 | 180 |
| Offspring fraction | 85 % | 75 % |
| Add Synapse Mutation | 2 % | 8.5 % |
| Add Neuron Mutation | 2 % | 12 % |
| Disable Synapse Mutation | 2 % | 19 % |
| Enable Synapse Mutation | 4 % | 6.9 % |
| Modify Weight Mutation | 80 % | 61 % |
| Modify Weight Perturb | 90 % | 47.1 % |
| Modify Weight Perturb-step | 10 % | 15.4 % |
| Crossover | 20 % | 32 % |

One of NEAT’s greatest strengths is its capability of protecting innovation (which only occurs through mutation). NEAT’s resilience to mutation through speciation is clearly seen in all 3 of the covered problems. The individuals obtained with these configurations performed well on all tasks. The XOR problem was correctly solved by them in 100% of the cases, the classification problem accuracy was ~90% and the Tic-Tac-Toe neural players learned to play perfect games 100% of the time.

Very high values for the mutation parameters can indicate many things; however the underlying purpose might not have been a best-fit for this problem. We wanted to see which combination of parameters leads to quicker convergence rates but a high mutation rate can only mean random search through the search space. This in turn may lead to a rather *deceivingly fast* convergence rate where in fact we are dealing with random search. The problem will be investigated further with another assumption and goal.

### Incestuous Crossover results

The results here were obtained by averaging 100 evolution cycles on each problem, separately. Table 6 shows the difference from using normal crossover to incestuous crossover in the XOR problem.

Table – XOR incestuous crossover results

|  |  |  |
| --- | --- | --- |
|  | Normal Crossover | Incestuous Crossover |
| Minimum Generation  Average Generation  Maximum Generation  Minimum # Neurons  Average # Neurons  Maximum # Neurons  Minimum # Synapses  Average # Synapses  Maximum # Synapses | 37  155  358  5  5.78  10  6  8.9  20 | 44  194  474  5  6.16  11  6  9.76  21 |

The classification problem’s incestuous crossover results in comparison with classical crossover are shown in Table 6.

Table – IRIS classification incestuous crossover results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Normal Crossover | Incestuous Crossover |  |  |
| Minimum Generation  Average Generation  Maximum Generation  Minimum # Neurons  Average # Neurons  Maximum # Neurons  Minimum # Synapses  Average # Synapses  Maximum # Synapses | 6  147  593  8  10.3  17  15  20.75  38 | 10  537  2450  8  13.35  25  15  26.95  56 |  |  |

Finally Table 8 will show the difference between the two types of crossover for the Tic-Tac-Toe problem.

Table – Tic-Tac-Toe incestuous crossover results

|  |  |  |
| --- | --- | --- |
|  | Normal Crossover | Incestuous Crossover |
| Minimum Generation  Average Generation  Maximum Generation  Minimum # Neurons  Average # Neurons  Maximum # Neurons  Minimum # Synapses  Average # Synapses  Maximum # Synapses | 39  372  860  19  21.2  27  90  94  105 | 121  749  2441  19  24.4  32  90  99.3  116 |

There are consistently worse results when it comes to both the average generation and the average number of synapses or neurons, thus staying true to the previous findings with regards to incest in genetic algorithms. The minimum number of generations required is also affected; however, we observe that the apparent minimal solution to these problems is found in both scenarios for all three environments. The minimum number of neurons and synapses for the XOR problem is indeed 5 respectively 6, and thus the crossover technique did not affect this; however, it consistently made progress slower and more cumbersome.

# Conclusions

We can see that NEAT is highly resilient to mutation because of its speciation, this being proved in our analysis using meta-NEAT. Moreover, we have seen a novel crossover approach which, intuitively should behave better than normal crossover, however our tests have proven that in fact, it actually hinders evolution and thus is to be avoided.

# Bibliography

1. *Evolving neural networks through augmenting topologies.* **Kenneth O. Stanley, Risto Miikkulainen.** 2002, Evolutionary Computation, pp. 99-127.

2. *A comparison between cellular encoding and direct encoding for genetic neural networks.* **Frédéric Gruau, Darrell Whitley, Larry Pyeatt.** Cambridge : MIT Press Cambridge, MA, 1996. pp. 81-89 .

3. *Solving Non-Markovian Control Tasks With Neuroevolution.* **Faustino J. Gomez, Risto Miikkulainen.** San Francisco : s.n., 1999.

4. *On Reinforcement Learning of Control Actions in Noisy and Non-Markovian Domains .* **Pendrith, Mark.** 1994.

5. *. Symbiotic Evolution of Neural Networks in Sequential Decision Tasks.* **Moriarty, David E.** Texas : s.n., 1997.

6. *Structural and Behavioral Evolution.* **Gregory M. Saunders, Peter J. Angeline, and Jordan B. Pollack.** Columbus : s.n., 1993.

7. *Incremental evolution of complex general behavior.* **Faustino Gomez, Risto Miikkulainen.** 5, 1997, Vol. Adaptive Behavior.

8. *Using Meta-Genetic Algorithms to tune parameters of Genetic Algorithms to find lowest energy Molecular Conformers.* **Zoe Ellen Brain, Matthew A Addicoat.** Odense : s.n., 2010.

9. *An evolutionary algorithm that constructs recurrent neural networks.* **P. J. Angeline, G. M. Saunders, J. B. Pollack.** 1, s.l. : IEEE, 1994, Vol. 5.

10. *Differential Evolution Training Algorithm for Feed-Forward Neural Networks.* **Jarmo Ilonen, Joni-Kristian Kamarainen, Jouni Lampinen.** 1, s.l. : Kluwer Academic Publishers, 2003, Vol. 17.

11. *Connectionist Theory Refinement: Genetically Searching the Space of Network Topologies.* **D. W. Opitz, J. W. Shavlik.** s.l. : Morgan Kaufmann Publishers, 1997, Journal of Artificial Intelligence Research, Vol. 6.

12. *Common Genetic Encoding for Both Direct and Indirect Encodings of Networks.* **Yohannes Kassahun, Mark Edgington, Jan Hendrik Metzen, Gerald Sommer and Frank Kirchner.** Bremen : ACM, 2007. 978-1-59593-697-4.

13. *Generating Large-Scale Neural Networks Through Discovering Geometric Regularities.* **Gauci, Stanley.** New York : ACM, 2007.

14. *A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks.* **Kenneth O. Stanley, David B. D'Ambrosio, Jason Gauci.** 2, s.l. : MIT Press, 2009, Artificial Life, Vol. 15, pp. 185-212.

15. *Measurement of the top-quark mass with dilepton events selected using neuroevolution at CDF.* **T Aaltonen, J Adelman, T Akimoto, S Zucchelli.** 15, 2009, Physical Review Letters, Vol. 102.

16. *Learning Bicycle Stunts.* **Tan, Jie and Gu, Yuting and Liu, C. Karen and Turk, Greg.** 4, s.l. : ACM, 2014, Vol. 33.

17. *Automatic Content Generation in the Galactic Arms Race Video Game.* **Erin Jonathan Hastings, Ratan K. Guha, Kenneth O. Stanley.** 4, s.l. : IEEE Transactions on Computational Intelligence and AI in Games, 2009, Vol. 1. 1943-068X.

18. **Jason Gauci, Kenneth O. Stanley.** Indirect Encoding of Neural Networks for Scalable Go. *Parallel Problem Solving from Nature.* s.l. : Springer Berlin Heidelberg, 2010.

19. **David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman.** Mastering the game of Go with deep neural networks and tree search. *Nature.* 2016.

20. *Functional Scaffolding for Composing.* **Amy K. Hoover, Paul A. Szerlip, Kenneth O. Stanley.** 4, s.l. : MIT Press, 2014, Computer Music Journal, Vol. 38.

21. *Reconciling Explanations for the Evolution of Evolvability.* **Bryan Wilder, Kenneth O. Stanley.** 3, London : Sage Publications, 2015, Adaptive Behavior, Vol. 23.

22. *Simple Evolutionary Optimization Can Rival Stochastic.* **Gregory Morse, Kenneth O. Stanley.** New York : ACM, 2016.

1. http://archive.ics.uci.edu/ml/datasets/Iris [↑](#footnote-ref-1)